National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

NL-339 (r. 88/04)

Canada

THE UNIVERSITY OF ALBERTA

An Analysis of a Broadband ISDN Signalling System's Physical, Data Link, and Network Layer Protocols for VLSI Implementation

by

© Jeffrey J. Brown

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH IN

PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL ENGINEERING

EDMONTON, ALBERTA

SPRING 1988

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-42746-9

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Jeffrey J. Brown

TITLE OF THESIS: An Analysis of a Broadband ISDN Signalling System's Physical, Data Link, and Network Layer Protocols for VLSI Implementation.

DEGREE: Master of Science

YEAR DEGREE GRANTED: 1988

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

#13, 1651 - 46 Street
Edmonton, Alberta, Canada
T6L 5H2

Date: Feb 18/88

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **An Analysis of a Broadband ISDN Signalling System's Physical, Data Link, and Network Layer Protocols for VLSI Implementation** submitted by **Jeffrey J. Brown** in partial fulfilment of the requirements for the degree of **Master of Science.**

W. Kinsner

Supervisor

A. R. Sebak

Emil Cienkowski

Date: 17 Feb. '88

Abstract

The thesis analyses the problems and potentials of implementing the physical, data link and network layers of B-ISDN-signalling systems in VLSI technology. It is proposed that the abundant bandwidth available on fiber optic media enables a new type of D channel to be used for B-ISDN systems. The proposed D channel adds overhead bits to every data byte to provide out of band packet framing and forward error correction. The improved functionality of the D channel permits an extremely simple data link layer to be used in place of LAPD since no packet retransmission procedures are required. In the network layer, the symmetric version of CCITT recommendations I.451 (Q.931) as it stood in August 1987 is analyzed. The network layer protocol processor is partitioned into a Protocol Specific Module (PSM) and an Application Specific Module (ASM). The PSM manages each call's state tables, timers and general call procedures. The ASM is responsible for interpreting signalling message content and configuring the communication equipment's user channel processing accordingly.

A major simplification to the call establishment procedures of I.451 is proposed, namely, the elimination of bearer channel negotiations using a two-way independent channel selection technique in which each end is master in the transmit direction and slave in the receive direction. The calling end of the link must specify in the Setup message the channel number(s) to be used in the calling-to-called direction. The called side specifies the called-to-calling channel number(s) in the Connect message. Additional simplifications accrue by not supporting stimulus terminal since they are incompatible with a symmetric protocol. The result is the elimination of several states in the call establishment procedures without loss of overall signalling or service functionality.

The partitioning, module interfaces, and functionality of the protocol processor is tested using a software simulation written in Pascal. The modified I.451 protocol is tested by executing the simulation on two computers connected via a simulated (RS-232) D-channel.

Acknowledgements

I would like to express my appreciation to the following organizations for their financial assistance:

- Alberta Telecommunications Research Centre
- Alberta Heritage Scholarship Fund
- Bell Northern Research
- Natural Sciences and Engineering Research Council of Canada
- University of Alberta

Appreciation is also extended to Dr. Witold Krzymien for his strong support and encouragement during the supervision of this work, the Alberta Telecommunications Research Centre for use of their excellent facilities, and Mr. Wayne Grover for his contribution during the definition stage of the project.

Finally, I thank my wife, Cori and my daughters, Ashley and Cassandra for their support and understanding throughout my graduate work.

Table of Contents

Abbreviations and Acronyms x

Chapter 1 Introduction 1

Chapter 2 Problem Description 3

2.1. Chapter Overview 3

2.2. Introduction to the ISDN Recommendations 3

2.3. Related Work 7

2.4. Definition of the Research Problem 8

2.5. Summary of the Research Performed 9

Chapter 3 Protocol Description 10

3.1. Chapter Overview 10

3.2. The System Model 11

3.3. The Physical Layer 14

3.4. The Data Link Layer 17

3.5. The Network Layer 24

3.5.1. Sources of Complexity in I.451 24

3.5.2. Call Establishment Options 26

3.5.3. Active Call Management 31

3.5.4. Call Disestablishment 32

3.6. The Protocol Subset 33

3.6.1. Protocol Processor Model 34

3.6.2. Error Handling Procedures 38

3.6.3. Call Establishment 42

3.6.4. Call Disestablishment 44

3.6.5. User Information Requests 45

3.6.6. Extending the System Model 45

Chapter 4 Protocol Processor Design 47

4.1. Chapter Overview 47

4.2. Timing Considerations 49

4.3. Interfaces Between Major Modules 52

4.4. Physical Layer Implementation 53

4.5. Data Link Layer Implementation 57

4.6. Network Layer Implementation 59

4.6.1. FIFO Based Message Flow Control 59

4.6.2. RxFifo Sub-module 63

4.6.3. TxFifo Sub-module 64
4.6.4. EPM Sub-module 66
4.6.4.1. The EPM-ASM Interface 66
4.6.4.2. The Event Queuing Technique 72
4.6.4.3. Functional Description of the EPM 76
4.6.4.4. Data Structures for Call Management 81
4.6.5. The ASM 82
Chapter 5 Software Simulation of the Protocol Processor 83
5.1. Chapter Overview 83
5.2. Benefits of a Functional Simulation 84
5.3. Structure of the Simulation Software 84
5.4. Hardware Simulation Control and Timing 87
5.5. The User Interface 90
5.6. Coding and Testing Procedures 91
5.7. Hardware Implementation: Is it Feasible? 93
Chapter 6 Conclusions and Future Work 96
6.1. Chapter Overview 96
6.2. Results Related to ISDN Standards 96
6.3. Feasibility of a Hardware Based Protocol Processor 97
6.4. The Simulation System 99
6.5. Future Work 99
References 101
Appendix A 103
Appendix B 109
Appendix C 114
Appendix D 115
Appendix E - Glossary 133

List of Tables

Table 3-1. Signalling System Model Signal Definitions....13
Table 4-1. L1-L2-L3 Signals Definition....55
Table 4-2. PSM Internal Signals Definition....62
Table 4-3. TxFifo Message Handling Definition....65
Table 4-4. EPM-ASM Interface Signals Definition....69
Table 4-5. Event Queuing Priority Table....74
Table 5-1. Simulation Software Module Definitions....86
Table 5-2. Simulation Software Data Flow Definitions....87
Table 5-3. Simulation Software Control Flow Definitions....87

List of Figures

Fig. 2-1. The Open System Interconnection Model 4
Fig. 2-2. Common Channel Signalling 5
Fig. 3-1. Point-to-Multipoint Signalling for 2B+D 12
Fig. 3-2. The ISDN Signalling System Model 13
Fig. 3-3. LAPD Packet Structure 18
Fig. 3-4. LAPD Address Field Format 18
Fig. 3-5. Communication Equipment Model 29
Fig. 3-6. ISDN Interface with a Time-Slot Interchange 30
Fig. 3-7. The Protocol Processor Model 36
Fig. 3-8. I.451 Message Structure 40
Fig. 3-9. Extended Network-Side Model 46
Fig. 3-10. Multiple Switch Connection Model 46
Fig. 4-1. Partitioning of the Protocol Processor Implementation 48
Fig. 4-2. Single Clock Handshaking 51
Fig. 4-3. Two Clock Handshaking 51
Fig. 4-4. L1-L2-L3 Interfaces 54
Fig. 4-5. L1->L2->L3 Data Flow Timing 56
Fig. 4-6. L3->L2->L1 Data Flow Timing 56
Fig. 4-7. PSM Partitioning into Sub-Modules 61
Fig. 4-8. The EPM-ASM Interface 68
Fig. 4-9. ASM-EPM Handshaking - Information Elements to Follow 70
Fig. 4-10. ASM-EPM Handshaking - No Information Elements to Follow 71
Fig. 4-11. ASM-EPM Handshaking - ASM Changes Its Mind 71
Fig. 4-12. ASM-EPM Handshaking - Illegal Request 71
Fig. 4-13. Signals Needed for Event Queuing 73
Fig. 5-1. Structure of the Simulation Software 85
Fig. 5-2. Inter-PC Wiring and Pinout 89

Abbreviations and Acronyms

2B+D	The ISDN basic rate interface consisting of 2 user channels and one signalling channel.
23B+D	The ISDN primary rate interface (North America) consisting of 23 user channels and one signalling channel.
30B+D	The ISDN primary rate interface (Europe) consisting of 30 user channels and one signalling channel.
ack	acknowledge
ASM	Application Specific Module
B-ISDN	Broadband Integrated Services Digital Network
BER	Bit Error Rate
BIOS	Basic Input-Output System
BNR	Bell-Northern Research
C/R	Command/Response field bit (LAPD)
call ref	call reference
CCITT	The International Telegraph and Telephone Consultative Committee
CCS	Common Channel Signalling
CCS#7	Common Channel Signalling system #7: a CCITT signalling system recommendation for inter-office signalling
CLK	clock
CPU	Central Processing Unit
CRC	Cyclic Redundancy Code
CTS	Clear To Send
DCE	Data Communications Equipment
DSR	Data Set Ready
DTE	Data Terminal Equipment
DTR	Data Terminal Ready
EA	Address field Extension bit (LAPD)
ECSA	Exchange Carriers Standards Association
EOM	End Of Message
EPM	Event Processing Module
FEC	Forward Error Correction
FIFO	First In-First Out data buffer
GND	ground
I.451	CCITT recommendation I.451 for the network layer
IC	Integrated Circuit

info	information
ISDN	Integrated Services Digital Network
Kb/s	Kilobits per second
L1	Layer 1 (physical layer) module
L2	Layer 2 (data link layer) module
L3	Layer 3 (network layer) module
LAN	Local Area Network
LAPB	Link Access Protocol - Balanced
LAPD	Link Access Protocol for the D channel
max	maximum
Mb/s	Megabits per second
MSc	Master of Science
msg	message
OSI	Open System Interconnection
PBX	Private Branch Exchange
PC	normally any personal computer, but in this case the IBM PC/AT
PSM	Protocol Specific Module
Q.931	CCITT recommendation for the network layer (identical to I.451)
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RTS	Request To Send
Rx	Receive
SAPI	Service Access Point Identifier (LAPD)
SDL	functional Specification and Description Language (CCITT recommendation Z.101-Z.104)
SOM	Start Of Message
TDM	Time Division Multiplex
TEI	Terminal Endpoint Identifier (LAPD)
TSI	Time-Slot Interchange
Tx	Transmit
µsec	micro-seconds
VLSI	Very Large Scale Integration

Chapter 1

Introduction

Integrated Services Digital Network (ISDN) is a network concept and a set of telecommunication interface recommendations developed by The International Telegraph and Telephone Consultative Committee (CCITT). The overall goal of ISDN is to create the frame work and standards necessary to permit public telecommunications networks throughout the world to evolve to the point where world wide fully digital end-to-end services are the norm. To help achieve this, the CCITT has specified a new family of digital communication protocols that are currently being considered by the numerous national standards bodies around the world. These protocols create the well defined interfaces needed to support standardized interconnection of communication equipment. To permit the use of different transmission media, several physical interfaces have been (or are being) defined. Although the early work on ISDN emphasized copper based media, especially the 2-wire twisted pair local loop, there has been much interest lately in the potential fiber optics has for supporting broadband services such as high speed data and full motion video. This area is commonly called broadband ISDN (B-ISDN).

A key aspect of all ISDN interfaces is the use of a sophisticated signalling system to perform call control procedures. The research reported by this thesis analyses the problems and potentials of implementing B-ISDN signalling systems when hardware is the preferred implementation technology. Although hardware implementation of the physical and data link layers is certainly not a novel concept, software intensive solutions are normally assumed for the network layer of the signalling system. A major thrust of this work is to explore the potentials and limitations of a more hardware oriented network layer implementation.

Chapter 2 begins with an outline of the general nature of ISDN interfaces, and specifies which of the ISDN protocols are of interest to the research. This is followed by a detailed definition of the research goals and an overview of the research performed.

In Chapter 3 an analysis of the functional requirements of the lower three layers of the signalling system for a hypothetical B-ISDN interface are presented. The discussion of the physical and data link layers emphasizes the utilization of the abundant bandwidth available on fiber optic media to simplify the overall complexity of the signalling system. The network layer discussion emphasizes reducing the complexity of the network layer signalling protocol to make it more amenable to hardware

implementation. To help with the analysis, models are developed for symmetric ISDN interfaces and for the signalling system's protocol processor.

In Chapter 4 the functional requirements defined in Chapter 3 are embellished considerably. The implementation aspects of the network layer are emphasized in this chapter since this is the area of greatest complexity both from a functional and an implementation point of view. The network layer portion of the protocol processor is partitioned into sub-modules, and the functionality and interfaces of each module are defined.

During the research, the functionality and partitioning of the protocol processor was analyzed and tested using a software simulation based on two personal computers establishing and disestablishing calls over a mock D channel. In chapter 5 the simulation technique and results are discussed.

Chapter 6 closes the thesis with a summary and interpretation of the research results. Those areas were further work is possible or required are also outlined.

Chapter 2

Problem Description

2.1. Chapter Overview

The goals of this chapter are:

- Outline the general nature of ISDN interfaces in order to create a simplified view of ISDN protocols. This provides the terminology and framework for later discussions.
- Demarcate the ISDN protocols (CCITT recommendations) of concern to the research described by this thesis.
- Discuss the state of the art in hardware implementation of ISDN and related protocols.
- State the hypotheses which directed the research and justify their validity as research topics.
- Outline the steps taken to test these hypotheses.

2.2. Introduction to the ISDN Recommendations

The last release of CCITT recommendations, the Red Book series, was published in 1985.[1] Since that time considerable enhancements have been added to the ISDN, especially in the network layer recommendations. The research described by this thesis is based on the 1985 Red Book I-Series Recommendations and the August 1987 version of Q.931/I.451 developed by CCITT working party XI/6.[2] The ISDN protocols created by the CCITT are called recommendations instead of standards because the CCITT is not a standards body. Adoption of CCITT recommendations is up to the national telecommunication standards organizations of the various member nations of the CCITT.

All ISDN protocols are based on the Open System Interconnection (OSI) seven layer reference model. The OSI model is essentially a set of commonly accepted structuring criteria that can be used in the design and specification of communication protocols. An example of a typical OSI model is shown in Figure 2-1. A full description of OSI can be found in the X.200 series of CCITT recommendations.[3] Note the dual nomenclature of the seven layers.

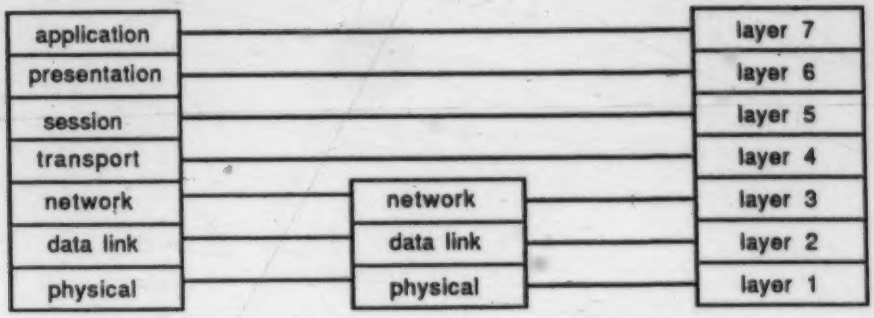


Fig. 2-1. The Open System Interconnection Model

The nature of the communication problem ISDN is meant to solve and the market segment ISDN addresses has shaped the type of communication protocols being grouped under the banner of ISDN. Digital transmission of voice and data between switching centers has been in place for several years, but the local loop - those last few kilometers of 2-wire twisted pair between the local switching office and the subscriber's home or business - is still almost exclusively analog technology.The CCITT's ISDN study group has worked at creating a fully digital interface protocol. What is meant by interface is a common boundary or reference point between the telecommunication network and the customer equipment. Although some consideration has been given to future access systems (such as fiber optic systems) the Red Book's I-Series protocols emphasize the existing copper based local loop.

ISDN defines 2 interfaces - basic access and primary access. Each is a time division multiplex of two types of channels: packet or circuit switched 64 Kb/s "B" channels, and a message (packet) based signalling channel called the "D" channel. Basic access is often called 2B+D since it consists of 2 "B" channels and a 16 Kb/s "D" channel plus some overhead. Primary access is either 23B+D (North America) or 30B+D (Europe) and consists of 23 (or 30) "B" channels and a 64 Kb/s "D" channel, plus overhead.

Besides the obvious differences in bit rate and number of B channels, the major difference between the basic rate and the primary rate interfaces is that a basic rate interface can be either point-to-point or point-to-multipoint depending on whether the interface is in the local loop or within the customer premise, while the primary rate is currently defined only for point-to-point connections. The point-to-multipoint basic rate interface has a bus topology designed for 4-wire twisted pair wiring. A maximum of eight drop points are allowed, with shared access to the 1 D and 2 B channels.

Both basic and primary access are structured such that the establishment of services over the B channels is controlled by the exchange of signalling messages over

the D channel. Therefore, the D channel is the "bit pipe" through which the ISDN's common channel signalling (CCS) protocol flows. This is shown in Figure 2-2.

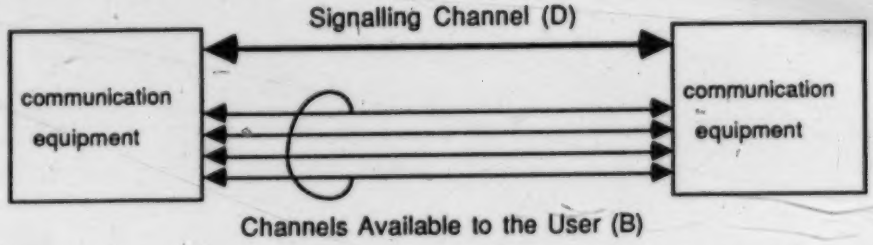


Fig. 2-2. Common Channel Signalling

ISDN's common channel signalling forms a separate multi-layered protocol embedded within the network layer of the overall access interface. This type of structure is difficult to show using the OSI model, but essentially it appears as a nesting of protocol definitions where the D channel is the physical layer of the ISDN signalling protocol, and the entire signalling protocol is the network layer of the access interface.

By definition, the network layer of a communication protocol is concerned with the establishment and routing of services through the network on an end-to-end basis. When the service to be established on a B channel is circuit switched voice communications the relationship between the ISDN signalling protocol running on the D channel and the service using the B channel is fairly clear. The initiating voice terminal (i.e. the calling telephone) exchanges messages on the D channel with the telecommunications network and a circuit (using one of the B channels) is set up between the calling and the called parties. When the called party answers the phone, the voice circuit is completed and the users can talk. Here, the signalling on the D channel supplies all the network layer functionality necessary to establish and disestablish the circuit switched service.

However, things can become confusing when the service to be established on the B channel is a packet service (for example X.25) in which the network layer of the packet service is embedded in the user's packets as they flow through the B channel. In this case, the D channel signalling is used only to establish and disestablish the bearer service (i.e. the "bit pipe") needed by the packet service to connect the user's terminal to the packet switch. The B channel can be thought of as the physical layer for the packet service. The packet service will use its own data link layer and network layer to actually control the flow and routing of the user's packets through the packet network. The B channel is like a "logical wire" connecting the packet terminal to the packet switching network.

So where has all this led us? Obviously it is very easy to become bogged down in details when working with the complete set of ISDN access protocols. The research presented in this thesis has concentrated on the D channel signalling protocols of ISDN since these protocols are new and their cost-effective implementation is critical to the success of ISDN. Therefore, what is needed is a simplified view of an ISDN communications link to use as a model in the analysis of the ISDN signalling protocols. Such a model is presented in section 3.2. For now, it is prudent to stop and state specifically which of the ISDN protocols we shall be concerned with.

The I.430 series of recommendations defines the physical layer for the basic and primary rate interfaces. The data link and higher layers of the ISDN signalling protocol assume very little about the interfaces physical layer, and actually assume very little about the D channel itself except that it is a duplex digital bit pipe running at some rate - normally 16 Kb/s or 64 Kb/s although any rate could be used.

The data link layer protocol is defined in the I.440 series of recommendations. The protocol is called Link Access Protocol for the D channel (LAPD), and it is a slight modification to LAPB, the data link layer used in X.25 type networks. LAPD can run in an "unacknowledged" mode, but the "acknowledged" mode of operation will probably be the standard. Those familiar with LAPB can appreciate that LAPD has a reasonably complex "packet retransmission on error" structure meant to compensate for the potentially high bit error rate of the local loop.

The network layer protocol is defined in the I.450 series of recommendations. The majority of work covered by this thesis is involved with Recommendation I.451: ISDN User-Network Interface Layer 3 Specification. The network layer is a very flexible message based protocol in which all signalling is performed via the exchange and forwarding of messages between and among network entities. Information within each message is grouped into information elements, which are in turn divided into information fields. Messages are structured by octets (bytes) and it is assumed layer 2 supplies the messages to the network layer in an error free, byte aligned fashion with the message boundaries already delineated. The network layer will receive much more attention later in the thesis.

Higher layers of D channel signalling are not yet fully defined by the CCITT, but there has been considerable work done to define the precise contents of the network layer messages needed for establishing and taking down the common B channel services such as voice, circuit switched data, and packet switched data. This work could probably be considered higher layer functionality, although its effect is best seen at the network

layer by the nature of the information elements that have already been defined for addressing, service specification, user-to-user message passing, etc.

The other recommendations in the I-Series cover some of the more general network, addressing (numbering plan), and service aspects of ISDN. These provide important background information and outline some of the general procedures necessary to establish a network-wide signalling functionality. Refer to Appendix C for a list of the relevant CCITT recommendations.

2.3. Related Work

The author is not aware of any publications that have dealt directly with ISDN network layer implementation issues to any real level of detail. The absence of publications is actually not surprising given that the vast majority of research and development in the field is currently being performed in industry on a proprietary basis. That is not to say that the topic of ISDN is not being widely discussed both in the trade journals and less technical publications such as the Wall Street Journal. A computer literature search performed by the University of Alberta in January 1987 located 2348 references to the terms ISDN or Integrated Services Digital Network, but no articles that dealt with implementation issues of the network layer. There is an article that discusses partitioning of the layer 2 and 3 functions[4] but the article does not contain detailed information concerning layer 3 implementation.

The concept of hardware implementation of the network layer is in fact a non-issue if current publications of IC manufacturers are to be believed. Papers submitted to the Wescon/86 conference[5] by major IC manufacturers were unanimous in declaring that the network layer would be implemented in software. A recent issue of BNR's technical journal, Telesis, was dedicated to ISDN, and the conclusion concerning network layer implementation in the DMS-100 switch was that the signalling would be mapped into an existing internal signalling format before being processed.[6] A survey of the data sheets and literature currently available from essentially all the major IC manufacturers producing ISDN components shows that, without exception, network layer processing has been delegated to some form of microprocessor based computing system. Companies such as Intel and Advanced Micro Devices have announced that they will be supplying software to perform some of the layer 2, and all of the layer 3 functions[7,8]. Given the fact that the ISDN network layer protocols are furthest away from standardization, it is no surprise that IC manufacturers are reluctant to commit the resources necessary to develop network layer ICs.

Because the ISDN signalling standards are very similar to the X.25 packet protocol standards, some useful background information can be obtained from texts which discuss networking in general, and the the X.25 standard in particular[9,10]. There have been many articles published concerning the various service, network, technical, regulatory, social and even political aspects of ISDN, but the best source of accurate and detailed information concerning ISDN signalling protocols is the CCITT Red Book supported by the more current publications of CCITT working party XI/6 (Digital Subscriber Line Signalling)[11]. The Exchange Carriers Standards Association (ECSA) is working on standardizing ISDN for the North American market. As of Spring 1988, the T1E1.4 working group is responsible for the ISDN access (physical layer) standards, with T1S1 concerning itself with the signalling aspects of ISDN[12]. An introduction to the general aspects of ISDN can be found in [Bartee][13] and [Miller][14], although in the latter the emphasis is on the physical layer.

2.4. Definition of the Research Problem

The research was performed to prove two hypotheses:

1) A powerful but simple subset of the I.451 protocol can be defined such that it is feasible to implement the network layer protocol processor in hardware.
2) The low bit error rate and large bandwidth capabilities of fiber optic based transmission systems can be exploited to simplify the implementation of ISDN signalling for Broadband ISDN (B-ISDN) interfaces at both the data link and network layers.

In the previous section it was noted that the current trend is toward a software implementation of the ISDN signalling system's network layer. The first hypothesis is based on a fundamental question: "Is software implementation of I.451 the only option?" Although it is true that the complexity of the fully featured I.451 protocol as it is defined today makes it a poor candidate for implementation in volume IC technology, there is an opportunity for VLSI technology to be applied to protocol subset implementations. The nature of I.451 is such that there are numerous subsets of the protocol that can be used depending on the particular communication service being implemented. Large reductions in protocol complexity can be made using this subset approach. Successful integration of some or all the the network layer functionality into a VLSI IC could result in the most cost effective implementation of layer 3 being hardware based.

As for the second hypothesis, CCITT recommendations for B-ISDN are not as advanced as for other ISDN interfaces so some assumptions had to be made. The suggested

implementation of the B-ISDN data link layer presented in this thesis is based on the assumption that the fundamental structure of B-ISDN interfaces will be similar to current ISDN interfaces: a single common channel signalling D channel controlling multiple user channels, all embedded in a TDM link. The goal is to show that by increasing the functionality of the physical layer to provide essentially error free transmission of signalling messages, a much simpler data link layer can be used for B-ISDN. The net effect would be an overall reduction in system complexity.

2.5. Summary of the Research Performed

The research being reported by this thesis consisted of the four stages described below. The stages are presented roughly in time order, although iterations between stages occurred throughout the research.

1) An analysis of the I.451 protocol, leading to the proposal for a network layer subset considered simple enough for hardware implementation.

2) An analysis of the physical layer and data link layer of fiber based B-ISDN CCS systems, leading to a proposal for novel physical and data link layer protocols for B-ISDN.

3) An analysis of the implementation considerations of the protocols developed in stages 1 & 2, resulting in the development of a suitable hardware partitioning and functional model.

4) The simulation of the protocols using two personal computers communicating over a RS-232 serial connection to simulate the D channel. The data link and network layer protocols where implemented in Pascal, with the software's functionality, modularity, inter-module interfaces, and signal flow timing all based on the hardware model developed in 3.

Chapter 3

Protocol Description

3.1. Chapter Overview

This chapter presents an analysis of the lower three layers of the signalling system for a hypothetical B-ISDN fiber optic interface. It is worth repeating that we are concerned with the OSI layers as they pertain to D channel signalling, not the user-to-user protocols that may or may not exist on services established over the B channels. The goal of the analysis varies depending on the layer being discussed.

The exact format and structure of the physical layer(s) that will eventually be defined by the CCITT and other standards bodies for B-ISDN interfaces is not a critical factor for this research. What is important is that B-ISDN systems based on fiber optic transmission systems will have an abundance of bandwidth and they can be engineered to guarantee a very low bit error rate (BER). This can and should be used to advantage by the D channel's signalling system. The goal of the physical layer section is to define exactly which properties of fiber optic transmission can be propagated to the D channel's physical layer and used to advantage by the other signalling layers.

The data link layer of B-ISDN signalling can be made much simpler than the current LAPD protocol simply because LAPD has greater functionality than is required for a fiber system. The data link layer section of this chapter addresses the second of the two research hypotheses presented in the previous chapter.

As one would expect from OSI based systems, the network layer defined by I.451 is quite portable to a B-ISDN interface. The key issues to be addressed in the network layer section pertain to the first of the research hypotheses. Unfortunately, the selection of an appropriate I.451 subset is somewhat dependent on the partitioning expected for the protocol processing hardware, and vice versa. Although the protocol subset is presented in this chapter, the full impact of the subset selected will not be seen until the hardware partitioning is discussed.

But first, it is helpful to create a model for a generic ISDN signalling system since the model will provide a common basis of discussion for the rest of the thesis, and hopefully eliminate some of the complexity that surrounds ISDN protocols. The model is developed in the first section, with the physical, data link, and network layer sections following.

3.2. The System Model

Because ISDN is normally considered an access protocol, the ISDN recommendations are based on a transmission link model in which one side of the interface is the "network side" and the other is the "user side". Anyone who has connected two computers together using RS-232 and fought with the DCE/DTE constraints of that protocol will understand the long term impact of not defining a symmetric communications protocol in which the communications equipment at both ends of the transmission link can operate as peers. Although symmetry was not addressed in the Red Book version of ISDN the most recent recommendations define a symmetric implementation of ISDN. Therefore, one constraint placed on our system model is that it be symmetric, or in other words, there should be no differentiation made between the ends of the transmission link.

Another desirable property of our signalling system model is that it be extendable to more than a single transmission link. ISDN is normally considered to be an access protocol only. The effort to define a symmetric version came from the realization that certain point-to-point applications such as a PBX-to-PBX link may wish to implement the ISDN standard. In general, it is assumed that the inter-switch signalling protocol called CCS#7 will be used with ISDN to create the network wide signalling system. However, it would be advantageous if our ISDN signalling system model was defined for a single transmission link but yet could be expanded to encompass a fully ISDN signalled network. Unfortunately it is premature at this point to discuss how the link model can be extended to model a network application. We will return to this point later in this chapter after the network layer protocol has been defined. For now we will stick to a signalling system model for a simple transmission link.

The next step in developing the model is to eliminate details that do not effect the true nature of the signalling protocol. An example of this is the bit rate of the D channel; it has no impact on the data link or network layer protocols. As well, the D channel need not share the same physical layer as the B channels as long as there is still a one-to-one correspondence between a D channel connecting the two ends of a link and a group of one or more B channels that also connect the link ends. Another way of wording this is that every B channel on a link is controlled by a pair of network layer protocol processors, each one responsible for one end of the link. These network layer processors communicate with each other over a single D channel using ISDN signalling protocols at the physical, data link and network layers.

Now, what about point-to-multipoint links? ISDN supports shared bus interfaces such as defined for 2B+D by creating a logically separate LAPD (data link layer) connection between the single ended node and each of the other nodes on the bus. Each

LAPD logical connection shares the single D channel, so the physical and data link layers create a D channel access protocol that resolves contention for the D channel across the multiple LAPD connections. However, the network layer and higher layers of the signalling protocol still treat their data link layer connection as a dedicated channel as shown in Figure 3-1.

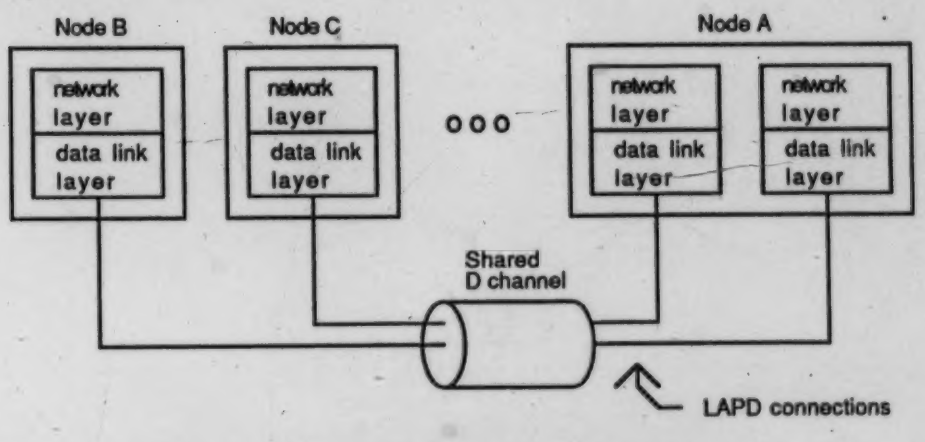


Fig. 3-1. Point-to-Multipoint Signalling for 2B+D

What still has to be resolved is whether or not a point-to-multipoint link can be modelled directly as multiple point-to-point links. The answer is yes, and no. Even though the network layers are not sharing LAPD connections and the D channel contention is resolved by the data link layer in a manner that is invisible to the network layer, the reality is that the various network layers must still cooperate when allocating the B channels. Using Figure 3-1 as an example, the AB network layer pair cannot allocate a B channel, say channel #1, without first making sure that the AC network layer pair is not already using channel 1. Fortunately, this level of knowledge - the allocation of B channels - is one of the network layer functions that can be kept out of the network layer interface and left up to the internal implementation of the nodes. The whole issue of allocation of functionality in the network layer is a subtle one, and is covered in section 3.6.1 - *Protocol Processor Model*. For now, let us assume the model can ignore point-to-multipoint links and leave the discussion of whether or not this is a valid decision until later.

The above discussions result in the model of Figure 3-2 being proposed. Table 3-1 describes the functional blocks of the above diagram.

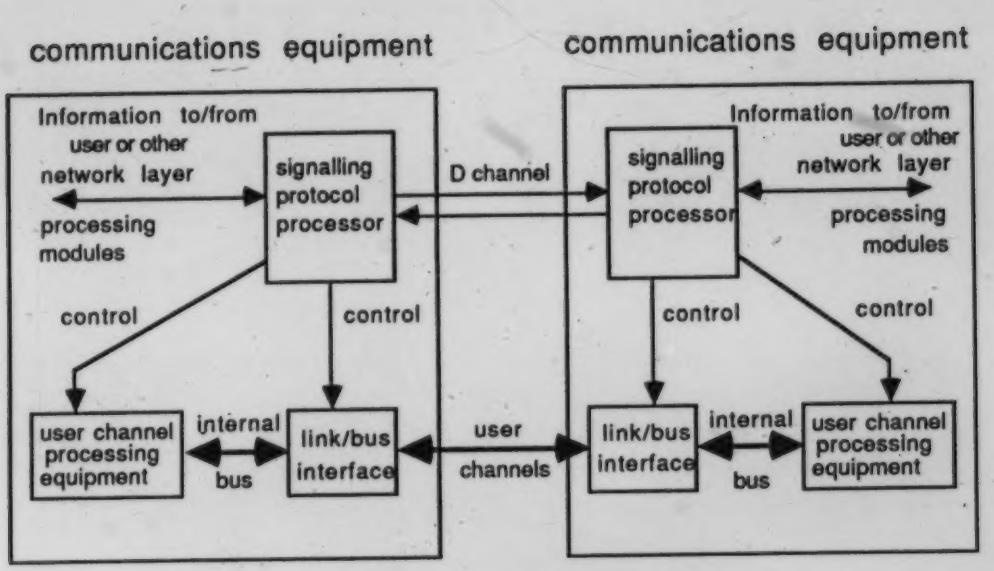


Fig. 3-2. The ISDN Signalling System Model

Functional Block	Description
Network Layer Protocol Processor	It processes all of the incoming and outgoing D channel messages, is responsible for the user channel allocations on the link, and controls the interface between the user channels and the internal bus of the communications equipment. It also makes and accepts signalling requests to/from user (if the node is a terminal) or other network layer processing modules (if the node is a switch for example). It communicates with the local user channel handling equipment so that signalling requests can be acted on.
Link/Bus Interface	Under the control of the network layer protocol processor it maps user channels on the communications link to/from the equipment's internal bus.
User Channel Processing Equipment	The type of user channel processing being performed by this equipment defines what type of node it is. In other words, this module is highly application specific.

Table 3-1. Signalling System Model Signal Definitions

In summary, the signalling system model is based on a point-to-point link with a single D channel carrying the signalling necessary to control the link's user channels. The communications equipment at each end of the link are peers, with each containing a protocol processor responsible for all signalling received and transmitted over the D channel. The model avoids the use of the term B channels intentionally because strictly speaking the definition of B channel is a 64 Kb/s channel. In the model, the channels available for user services do not have a defined bandwidth, although it is assumed the user channels are a Time Division Multiplex (TDM) of circuit switched bearer services available for the user's circuit or packet switched teleservices.

The model is embellished considerably when the partitioning of the protocol processor is discussed, but first an analysis of the lower three layers of the proposed B-ISDN signalling protocols is presented.

3.3. The Physical Layer

Although the CCITT's work in B-ISDN is ongoing, everyone agrees there will eventually be a B-ISDN recommendation for fiber based interfaces. The precise details of the future interface specification are actually not that important for this discussion. What is important is that there are some significant differences between transmission over fiber and transmission over twisted pairs. Both theoretically and practically, fiber optic transmission systems can provide a much lower "cost per transmitted bit" and a much lower BER than copper based systems. These differences should be taken into account when designing a CCS system for B-ISDN. Because the physical layer of CCS is the D channel itself, the discussion will center on the properties and functionality a B-ISDN D channel can provide. The size (bandwidth) of the D channel is addressed first, followed by a discussion of the D channel's BER. The section culminates in a description of the physical layer functionality assumed by the higher layers of the proposed B-ISDN signalling protocol.

In B-ISDN, the bandwidth consumed by a 16 Kb/s or 64 Kb/s D channel is no longer a critical resource. With 2B+D and 23B+D the bandwidth consumed by the D channel is a significant portion of the total bandwidth of the interface. For comparison, if a 150 Mb/s B-ISDN interface had a D channel sized in the same user channel to D channel bandwidth ratio as 23B+D then the B-ISDN D channel would have a bandwidth of over 6 Mb/s! Clearly there is no bandwidth based argument for a 16 Kb/s or 64 Kb/s D channel for B-ISDN. Some may argue that B-ISDN's D channel should remain 64 Kb/s in order to take advantage of the hardware developed for the primary rate interfaces. This argument is flawed in two areas: it assumes primary rate hardware is (or will be) plentiful and cheap, and it assumes the D channel processing hardware/software is portable to B-ISDN equipment. Neither assumption is necessarily valid.

But if we take the lid off the D channel's bandwidth restrictions, how large should the D channel be? If the D channel bandwidth is inordinately high, then expensive high speed D channel processing equipment will be needed at every B-ISDN node. If the D channel is too slow then its information carrying capabilities are diminished unnecessarily which could reduce the suitability of B-ISDN interfaces to high activity applications such as PBX-to-PBX connection. What should be done is to characterize prototype implementations of D channel processors in order to determine the D channel

bandwidth that can be handled cost effectively. This thesis is a first step to this type of result.

Besides available bandwidth, the other aspect of fiber optic systems that can impact the design of the B-ISDN D channel is the low BER fiber systems exhibit. One often hears of fiber systems designed to meet a BER requirement of less than 10^{-9} . With numbers such as these, one is tempted to question whether fiber optic B-ISDN systems will really require error correcting techniques applied in the data link layer. Given that the bandwidth of the D channel is not an overly expensive resource in a B-ISDN system, why not apply forward error correction (FEC) to the D channel to guarantee a BER that is so low errors on the D channel are orders of magnitude less likely to occur than errors created by the signalling protocol processor itself. Further, since some FEC codes also detect (but do not correct) certain multiple error conditions, the D channel could notify the upper layers when an error has occurred but no corrective action is possible.

It is not the goal of this thesis to probe the detailed nature of errors in fiber systems or to determine the type of FEC necessary to make the inherently low probability of errors even lower. Rather, it is assumed the physical characteristics of the D channel can be engineered such that there is no need for error detection techniques in the data link layer. This is an extreme stance, so one might be tempted to back off to the point where the link layer at least applies a message oriented error detection mechanism such as a cyclic redundancy code (CRC). As will become clear later, use of a CRC is a fairly simple extension to the system architecture, even though no CRC is used in the initial design.

To summarize, we assume the B-ISDN D channel provides a fairly high bandwidth channel which runs essentially error free, although certain detectable error conditions can occur. Of course no correction or detection system is perfect so there is always the chance that an error will get through uncorrected and even undetected. The assumption is that this event is so unlikely we can ignore it since other failures (such as software failures) are far more likely to corrupt the system's signalling system.

One might argue that applying FEC to the D channel is actually a layer 2 function and therefore its definition belongs in that layer. Another way of looking at it is that the physical layer has been improved to the point were one of the functions normally placed in layer 2 is no longer required, and hence can be eliminated. Whether FEC is called a physical layer or a data link layer function the key point is that the intrinsic properties of fiber transmission were used to provide a required function - the error free transmission of information across the link - in a novel and simple manner.

The obvious question now is whether there are any other signalling functions that an improved physical layer could help provide in a way not feasible on ISDN systems based on twisted pair technology. It was noted earlier that the I.451 network layer not only expects error free messages from the layer 2, but the messages are to be byte aligned, with the start of message (SOM) and end of message (EOM) already delimited. LAPD adds its own packet framing to the raw bit stream provided by the D channel by using a 01111110 flag sequence to mark SOM & EOM, with 0 bit insertion (stuffing) used in the rest of the packet to prevent any 6 bit long sequences of 1's occurring in the packet's data. While this scheme is not that complex, there is a more straight forward way of delineating packets.

LAPD uses the flag framing technique because one cannot restrict the contents of the 8 bit wide D channel, and hence one cannot define a unique 8 bit SOM pattern. But why restrict the D channel to 8 bits wide? If we made it 9 bits wide, for example, the extra bit could be set to a 1 when valid data exists in the other 8 bits, and to a 0 otherwise. Not only is the need for 0 bit insertion eliminated, there is no longer any need for the extra SOM and EOM flags. This increases the D channel bandwidth requirement by 12%, but is that really of concern on a 150 Mb/s B-ISDN interface? Since many of the messages used by the I.451 protocol are only a few bytes long, the elimination of a start flag at the beginning of each packet is actually significant, so some of the bandwidth "wasted" by the 9th bit is recovered eventually.

Further, it is easy to envision using extra bits for more than just marking SOM and EOM. Perhaps extra bits could be added for a block code FEC scheme that also marks SOM and EOM. At this stage it is not so much the details that are important but rather the concept of using the "cheap" bandwidth of fiber to simplify higher layers of the CCS system.

For our protocol simulation, it is assumed the SOM and EOM are marked using extra bandwidth on the D channel. This is actually not a critical part of the research, since a LAPD type of D channel packet framing could have just as easily been used. The only impact on the design would be that the timing between layers 2 and 3 would have to allow for the extra framing flags added by layer 2 to mark SOM and EOM.

In conclusion, the data link and network layers of the proposed B-ISDN signalling system assume the D channel has the following properties:

1) The probability of undetected errors occurring on the D channel is low enough to be ignored.

2) The physical layer is capable of recognizing certain uncorrectable errors and reporting these to the higher layers. The probability of these detected errors

occurring is also extremely low (say 1 erroneous message per month of continuous operation).

3) The D channel provides the "per byte" overhead required to mark SOM, message contents, and EOM without adding extra bytes to the front or back of the packets being sent.

4) The combination of points 2 and 3 results in the physical layer classifying every byte on the D channel as one of (<empty>, <som>, <message contents>, <eom>, <errored>).

A detailed definition of the functionality and interface provided by the D channel is given in the physical layer interface section later in the thesis. For now, the general description given above is sufficient to discuss the functionality of the proposed data link layer.

3.4. The Data Link Layer

The physical layer functionality defined above eliminates the need for two key functions of the LAPD protocol - error correction via packet retransmission and packet delineation using framing flags. However, there are several other layer 2 requirements that need to be discussed to determine how they may be implemented in a B-ISDN system. The section begins with an analysis of the remaining functionality required of layer 2. This is followed by a summary of the layer 2 capabilities being assumed by the network layer.

When the network layer wishes to send a message out on the D channel, it passes the message to the data link layer which adds the SOM flag, two address bytes and two control bytes to the beginning of the message, and a two byte CRC and EOM flag to end of the message, thereby creating a LAPD packet. The packet is then passed to the physical layer for transmission. At the receive side, layer 2 receives the packet from layer 1, checks the CRC to make sure the packet has been transmitted error free, strips off the rest of the overhead added by layer 2 at the transmit side, and passes the message to the network layer entity addressed by the packet. An example of the packet structure used by LAPD is given in Figure 3-3. It should be noted that there are control packets generated by LAPD that do not contain any network layer information field. The structure of these packets is slightly different, but that can be ignored for now.

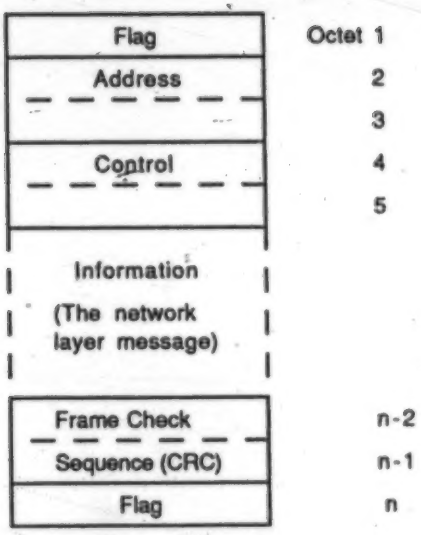
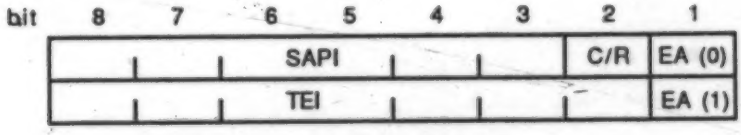


Fig. 3-3. LAPD Packet Structure

Each field added to the network layer message by LAPD exists to satisfy one or more of the functional requirements expected of an ISDN data link layer. The SOM flag and EOM flag have already been discussed. In our layer 2 proposal these fields are replaced by added functionality in the physical layer. But what are the roles of the other fields?

The address field is further defined by LAPD as follows:



EA Address field extention bit
C/R Command/response field bit
SAPI Service access point identifier
TEI Terminal endpoint identifier

Fig. 3-4. LAPD Address Field Format

The EA bit is there to permit future extensions of the addressing to multiple bytes, while the C/R bit is part of the LAPD control protocol. The fields of real interest are the two addressing fields. Note that these are LAPD addresses, so they have no significance outside the data link layer.

First the TEI. In a point-to-multipoint ISDN link, the network layers of each of the terminals on the multidrop side of the link require their own virtual data link layer connection to a peer layer 3 entity in the single ended side of the link. In the basic rate interface standard, the D channel is shared by all terminals on the link, so layer 2 must

have some way of uniquely identifying each of these logical layer 2 connections. This is the purpose of the TEI, it identifies which logical connection the packet belongs to. Since all terminals share the D channel on a broadcast basis, the TEI is used by each terminal to determine if the packet it has received is actually for it, or for one of the other terminals on the link. There are three classes of TEI addresses. TEI values 0-63 are available for terminals to use on a fixed or hardwired basis. Values 64-126 are assigned dynamically using an assignment algorithm defined by LAPD. TEI 127 is the address used to broadcast a packet to all attached terminals. All TEIs active on a link must be unique.

The SAPI address field is complementary to the TEI address. The SAPI is used to address a particular type of network layer within the terminal addressed by the TEI. In the Red Book version of LAPD, only three SAPI addresses are defined, 0 for normal call control, 16 for user packet communications, and 63 for management procedures. The packet communications SAPI is defined because the CCITT did not wish to rule out the possibility that the D channel could also be made available to the user as a packet service. Management procedures are not defined by the Red Book, but the idea is that most telephone operating companies badly need better control over the operations, trouble shooting and maintenance of their equipment. Having an "intelligent" signalling channel used to remotely poll and/or command equipment is seen as an important feature of ISDN networks.

Basically, that is LAPD addressing. We have intentionally left out some details to avoid confusing things since the real question to be answered is: do we really need all this data link layer complexity in a B-ISDN signalling system? Addressing at the data link layer is essentially the multiplexing of logically separate network layer communication channels onto a single D channel. It boils down to two points:

1) Will the link contain multiple terminals/nodes sharing a single D channel in a bussed configuration?

2) Will there be more than one network layer entity per terminal/node?

If we look back at our signalling system model from section 3.2, the answer to both questions is no. But is the model general enough to be valid for the potentially large number of very diverse applications that will fit under the umbrella of B-ISDN? Certainly we cannot rule out for all time the existence of multidrop B-ISDN busses, nor can we rule out the need for a maintenance channel. As for user access to the signalling channel for packet services, it seems strange for a B-ISDN interface to try to save the bandwidth required for a separate packet service channel by allowing the user access to the D channel, but if the argument to do so is based on equipment sharing as opposed to bandwidth sharing one can envision a scenario in which users would benefit from

occasional packet service access on the D channel. So what good is a signalling system model in which there is only one signalling path per link, especially when the discussion turns to layer 2 instead of layer 3?

There are two ways to look at this problem. The first approach is to assume that since bandwidth on a B-ISDN link is so cheap it is not necessary to multiplex multiple layer 2 connections onto a single D channel. Each layer 2 connection can have its own D channel, with the multiple D channels being multiplexed into the TDM interface at layer 1. In this scenario, the model is completely accurate as long as the various signalling connections are mutually independent. Actually, this approach is not as drastic as it sounds. Even with a multiplexed data link layer, each of the signalling connections requires its own network layer pair, so there is no added expense at the network layer if multiple D channels are used. As well, since layer 2 will be extremely simple if we throw out the ability to support multiple layer 2 connections over a single D channel, it is not a heavy penalty to require multiple independent data link layers to be supported instead of a single multiplexing version. Finally, since the hardware required to TDM several D channels into the B-ISDN interface is very simple, the physical layer of the signalling system will not be overly complicated by multiple D channels. Ultimately the use of multiple D channels comes down to an argument between bandwidth cost versus processing complexity cost. Such an argument is not easy to resolve.

The second approach to the modelling problem is to avoid the whole issue by admitting the model is only valid for single layer 2 connections, and that an extension to the model will be required if a multiplexing data link layer is to be considered. This is the approach used in the research. Basically, the assumption has been that if network layers other than the one needed for call control on a point-to-point link are needed, they can be accommodated either via a multiplexing data link layer or by using multiple D channels. This is an area that will have to be addressed in later research.

So in conclusion, for the simulation we assumed layer 2 does not need to prefix the layer 3 message with addressing fields.

Now for the control field of LAPD packets. The key function of the control field is to permit the cooperating LAPD processors at each end of the link to exchange the information necessary to keep the complex LAPD protocol working. It has already been stated that the data link layer need not correct for errors since their probability is very low. Therefore, there is no need for the cooperating data link layers to exchange information concerning the retransmission of network layer messages. The question that remains is whether or not there is a need for the data link protocol to exchange layer 2

to layer 2 information for other reasons. Two possibilities are flow control and link establishment.

Flow control in layer 2 might be needed for several reasons:

1) The layer 2 processing entities at the two ends are implemented differently, and one end can "swamp" the other end with too many packets.

2) The network layer is incapable of accepting packets as quickly as the data link layer can provide them. This can be solved either by local buffering of the packets, or by temporarily slowing down or stopping the transmission of packets at the far end.

3) A hardware or system failure has taken layer 2 out of service.

The third point is best treated as a link establishment problem. The second point is solved either by a purely local solution or by reducing the problem to the first point, namely, slowing down the far end. Since it is impossible to predict the implementation technology that will be used at every B-ISDN node, one might be tempted to conclude that some form of flow control will indeed be necessary. However, a better approach is to look at the potential source of the processing mismatch because, as it turns out, the problem will actually never occur!

So far layer 2 does not really do anything to the packet as it comes in. Later in this section we will show that layer 2 will actually do some processing (i.e. it is not a totally null layer), but fortunately this processing is done on a byte by byte basis. Therefore, there is no reason why the packet could not be passed up to the network layer byte by byte as it arrives. Our problem then becomes one of flow control in the network layer, not the data link layer. As will be shown, I.451 has a natural flow control built into the handshaking protocol. Therefore, flow control at layer 2 is a non-issue as long as the interface to the network layer is capable of accepting packets on a byte-by-byte basis at the rate they arrive from the physical layer. This is described fully in Chapter 4..

So much for flow control, what about link establishment. In LAPD there are several modes or states of operation defined. One state exists before the TEI is assigned to the link. Another state exists before multiple frame operation (used for packet retransmission buffering) is established. There is also the potential for a "low power" mode in which the link is only partially active during periods of no activity. The first two states will not be needed for this implementation. However, the low power state and the overall question of establishing link integrity at power up or after a hardware or D channel failure is still at issue.

There are many areas to cover when discussing link establishment. A fundamental question to answer before communications of any type can be established is how do I

know when the far end is ready? In this case, because the D channel is bi-directional and operates continuously, the physical layer can notify all higher layers when the far end is powered up and ready to receive. Further, since we have only defined one mode of operation for the data link layer it is fairly easy to specify that the data link layer must be ready before the physical layer becomes active. As long as layer 2 has only one active state -ready to receive- there is actually no need for link establishment procedures at layer 2. Is this one state enough though? If there is no TEI assignment required and low power mode is handled at the physical layer by shutting down the D channel, then the only other problem is failure of the layer 2 processor itself. In this case there is not much that can be done at layer 2 anyway, after all it has failed, so it must be up to the physical layer to notify the far end of the failure. Since for all practical purposes this is equivalent to a failure of the D channel itself (from the CCS system's point of view at least) there really is only one *active* state for the data link layer defined here.

So in conclusion, as long as we keep dynamic TEI assignment out of layer 2, the state of the data link layer is directly reflected by the state of the D channel. Either they are both active and working, or they are both dead. Hence, there is no need for peer level data link layer information exchange other than for the exchange of network layer messages. Therefore, no control field is required.

And now finally the frame check sequence or CRC field of a LAPD packet. It was mentioned in the physical layer section that a CRC could be added by layer 2 if the physical layer's ability to detect errors on a byte-by-byte basis was deemed insufficient. This warrants some further discussion. Notice that the phrase used was on a "byte-by-byte" basis and not "block-by-block". This is important because the physical layer of most interfaces (i.e. the full B-ISDN physical layer including user channels) will provide some form of error detection such as a CRC run over the previous frame. At the low error rates expected of fiber links one might be tempted to assume that whenever an error occurs somewhere on the link, the D channel is suspect and hence the contents of the D channel in the previous frame should be classified as erroneous. Besides causing unnecessary "hits" on the D channel, this technique forces a 125 μsec delay in the processing of D channel messages. Even though a message may have been received completely in one frame, no actions can be confirmed until the next frame's error check is validated. This is no problem if the D channel is only 64 Kb/s, but in a 2 Mb/s D channel (not an unreasonable number for a 150 Mb/s B-ISDN link) this delay is probably unacceptable.

Therefore, if the D channel is to support some form of error detection over and above any FEC being applied, the detection should be applied on a byte-by-byte basis.

Unfortunately, this restricts the use of FEC block codes to those which apply to a block size of 8 bits or less. Of course larger blocks could be used, but their ability to report some multiple bit errors cannot be used. The penalty of using smaller block sizes for these types of codes is, of course, bandwidth since generally the ratio of added bits to data bits decreases as the block size increases. Since the D channel is probably not a major portion of the overall bandwidth of a B-ISDN link, even doubling the D channel's bandwidth in order to improve its BER may be quite acceptable.

But even if FEC is applied to every byte of information passed through the D channel, there is still a chance that a multiple bit error will corrupt one or more bytes in a manner that is undetectable by the FEC scheme used. What can be done about these errors? When the base BER of a link with FEC is 10^{-9} it is tempting to ignore these errors at layers 1 and 2 since the probability of them occurring is so remote after FEC is applied. For example, even if the D channel can be guaranteed to have an undetected error rate of no greater than 10^{-14} (after FEC) then a 1 Mb/s D channel will, on average, pass undetected less than one errored bit per thousand days.

Even after all this is said, it is so important that the CCS system remains operational even if the overall link is suffering from a major but not fully disabling degradation of error performance it still might be desirable for the data link layer to apply a CRC to each network layer message. The reasoning behind this is that it is far worse for an errored packet to get to the network layer undetected than to detect the error in layer 2 and simply tell layer 3 to ignore the packet it has just received. The problem with a CRC is that the error cannot be detected until the end of the message is received, and this might prevent the network layer from processing the message byte-by-byte as it is being received. As will be shown, the network layer needs the ability to "unwind" the actions it has initiated as it receives a message anyway, so the addition of a CRC by the data link layer will not adversely hamper the network layer's implementation.

In summary, although no CRC was added by layer 2 for the simulated B-ISDN CCS protocol, it would be a straight forward extension to the data link layer to do so.

We have successfully eliminated all the extra bytes added to layer 3 messages by LAPD. Does this mean the data link layer we are proposing is a totally null layer? The answer is no because an ISDN data link layer such as LAPD does more than just pass bytes from layer 1 to layer 3 and from layer 3 to layer 1. The data link layer interfaces to the network layer at the message level, where a message is a group of bytes delineated by a SOM and EOM. Further, since the minimum length of an I.451 message is 4 bytes, the data link layer can apply a few simple tests to the bytes it receives from the physical

layer to insure they do actually form a packet with a SOM, non empty message contents, and a EOM. It can apply these tests with no knowledge of the actual contents of any of the bytes since the physical layer we have defined provides an out-of-band classification of each byte as one of (<empty>, <som>, <message contents>, <eom>, <errored>).

The details of the layer 2 algorithm necessary to test for overall packet integrity is described fully in the next chapter. The net effect of this layer 2 functionality on the implementation of layer 3 is that while layer 2 is in the process of passing layer 3 a message there is a possibility layer 2 will notify layer 3 that the message is errored and should be ignored. This forces the implementation of the network layer to hold off confirming any actions initiated by the receipt of the information elements of an incoming message until after layer 2 has notified layer 3 that the whole message has been received, and no problems (including a possible CRC error) have been detected. This restriction is discussed fully in the next chapter.

3.5. The Network Layer

This section describes the nature of I.451's complexity and where this complexity can be reduced. The section opens with a discussion of the general sources of complexity in I.451. This is followed by three subsections focusing on how I.451 can be simplified in the establishment, active, and disestablishment phases of a call. The next section provides the detailed description of the subset I.451 protocol selected for the simulation.

3.5.1. Sources of Complexity in I.451

If there is one place where ISDN shows its heritage most it has to be in recommendation I.451. The difficulties in creating a universal signalling protocol to meet the access needs of a world wide telecommunications network are evident in the impressive flexibility built into the I.451 protocol. Unfortunately this flexibility also adds an impressive amount of complexity to the protocol. Even a cursory scan of the I.451 recommendation leaves one with a solid conviction that actually trying to implement the protocol is not for the faint of heart. And the last thing that comes to mind is the urge to implement it in hardware!

The I.451 recommendation consists essentially of three things:

1) A definition of the permitted signalling messages. The definition covers the structure, content and the actual coding into 1's and 0's of all messages, information elements, and information fields.
2) The definition of the handshaking protocol required to control call set up, modification, and take down. The protocol takes the form of a state machine

implemented on a per call basis by each end of the link. The state machine is defined both in tabular and graphical form. The graphical form is the standard CCITT method called functional Specification and Description Language (SDL). It is the SDL diagrams that best reflect the overall nature and complexity of the signalling protocol. Appendix B contains a simplified version of the August 1987 I.451 SDL diagram for the user side. The network side SDL diagram is significantly more complex than the user side SDL diagram, but as is shown in the last section of this chapter, a symmetric protocol based on the user side protocol can be extended to cover the network side's requirements.

3) A set of textual specifications that embellish the SDL diagrams and explain what actions should be taken at each call state. These are called the "call control procedures". They help in the interpretation of the SDL diagrams and define how to handle error conditions not covered by the SDL diagram.

The messaging structure adopted for I.451, although extremely flexible, is very regular. This makes parsing of messages, information elements and information fields relatively straight forward. The problem is not extracting the signalling information from the messages, but rather what to do with the information as it arrives. Because of the overall complexity of the I.451 handshaking, there are numerous ways messages can be exchanged and information passed to achieve the same net results. This appears in the SDL diagrams as multiple paths between states. It is this complexity that is most worrisome to a hardware implementation since there is an overwhelming number of options that must be handled at almost every stage of the call set up and take down procedures.

But why do all these ways to achieve the same thing exist? Certainly one of the biggest reasons was the need to support user terminals of widely varying capabilities. CCITT defines two types of terminals: "stimulus" terminals and "en bloc" terminals. The difference between the two is that stimulus terminals do not remember what their call state is - they are "dumb". The idea is that ISDN will not be accepted unless a simple ISDN handset can be made available for about the same price as today's analog telephones, and unless the overall cost of ISDN access is about the same as it is today. The reasoning behind stimulus terminals is that if the network has all the "smarts" then terminal costs are low and equipment can be shared at the local switch.

The problem is that the need to support these dumb terminals has greatly complicated the I.451 protocol and essentially eliminated the possibility that I.451 could be a symmetric protocol. Clearly stimulus terminals are slaves to the network side of the interface. We could at this point enter a rather lengthy discussion of whether or

not hiding extra smarts in the network side of the interface will actually reduce the cost of ISDN services to the home. The argument is that once ISDN terminals are mass produced the cost of stimulus terminals will be essentially identical to simple en bloc terminals. But instead of being side tracked we can simply bypass the issue by assuming at the start that all terminals or any other pieces of equipment attached to the interface must be intelligent enough to fully support the network layer subset selected. After all, the goal of the research is to prove that I.451 can be made simple enough to be implemented in hardware, and hopefully it is hardware implementation of all three signalling layers that will prove to be the lowest cost solution.

Support of stimulus terminals, while an important source of complexity for I.451, is certainly not the only area that should be analyzed for potential simplification. The trick is to walk that fine line between adding too little and adding too much. The CCITT has a target market in mind for basic rate and primary rate ISDN. That market is international and highly diverse. If we make the assumption that people will want to use ISDN signalling protocols in environments that are not so difficult to design to, for example the PBX and LAN markets, then it is tempting to see if walking a different line than the one taken by the CCITT will lead to a less complex I.451 protocol. Further, since B-ISDN is essentially free from the constraints of history there is additional justification for re-examining the I.451 protocol for the B-ISDN market.

3.5.2. Call Establishment Options

It is obvious from the SDL diagrams of I.451 that the call establishment phase contributes greatly to the complexity of the network layer protocol. This subsection discusses the call set up portion of the I.451 protocol with the goal of retaining the overall power of the full protocol but yet eliminating those options that are limited to special applications, or are simply redundant.

The Red Book defines seven messages for call establishment. They are Setup, Setup Acknowledge, Information, Call Proceeding, Alerting, Connect and Connect Acknowledge. Although Setup, Connect and Connect Acknowledge are the only messages needed for a minimal system, the other messages are used to add features or flexibility to the call establishment procedure. In general, the various call establishment messages are used as follows:

Setup - This message begins every call establishment. It notifies the destination of the type of service required (the Bearer Service element), the destination and source addresses (if required), and possibly the channel number(s) to be used for the connection.

Setup Acknowledge - This message is used by the destination end to reply to the Setup message. It is used to permit the channel identification to be negotiated between the two terminals, as is described later.

Information - This message is used in response to the Setup Acknowledge message. It continues the channel negotiations from the source end.

Call Proceeding - This message was intended to be used by local office switches to effectively place the call establishment "on hold" until a response from the destination of the call can be reached. This is needed because the switch is really only the intermediary between the source and destination of the call. In general, however, Call Proceeding can be used by any equipment that needs some time before a response to the Setup message can be made.

Alerting - Once the channel identification is known, the call can proceed. Unfortunately, when humans are involved just because a channel is available does not mean that the call can actually be completed. The Alerting message is used to notify the source that the destination has received the call and is currently notifying the user that the call is present. This is the call state when the phone is ringing. However, there is actually no reason why the Alerting message cannot be used by data equipment that is waiting for a resource to become available.

Connect - The Connect message is sent from the destination to the source to let it know that the call has been received and can proceed. The destination must be connected to the bearer channels (i.e. a working connection must be established) before it sends this message.

Connect Acknowledge - This message is used by the source to confirm receipt of the connect message. It completes the handshaking between source and destination. The source must be connected to the bearer channels before sending this message.

There is at least one new message of concern to the call establishment procedures that has been added to the protocol since the Red Book. It is called the Progress message and it is used to relay additional network interworking and inband tone information. Because its use is not firmly defined it has been eliminated from the protocol.

Whether or not a particular call establishment requires the Alerting message depends on the type of service that call is establishing. Calls directly between machines may not require the Alerting message; the protocol could step directly to the connect phase. However, human-to-human calls such as voice and video will need the Alerting message since humans are relatively slow to answer their calls, and the calling party likes to know whether the called party's phone is ringing.

But what about the Setup Acknowledge and Information messages? Their use depends ultimately on the type of bearer channel selection technique used by the link. The various channel selection options available in I.451 and the impact of the additional requirement for a symmetric protocol on the bearer channel selection process are discussed next.

A side effect of turning I.451 into a symmetric protocol is that the bearer channel negotiations must also be symmetric. In the normal network-user protocol, the network (or the user terminal for that matter) could unambiguously assume the role of master when channel negotiations were needed. In a symmetric protocol, there is always the danger that both ends will try to simultaneously seize the same channel(s) while attempting to originate a call. In Annex D "Extensions for symmetric call operations" of the August 1987 version of I.451 the procedures for handling call collision are defined. Unfortunately they require the temporary assignment of one side as the "network" and one side as the "user", thereby eliminating true symmetry. The algorithm for determining which channels are assigned to each of the calls is also quite complex in order to handle the various ways I.451 permits the bearer channel selection to be negotiated between the network and the user sides. Further, the method used by each side to determine just which is the network side and which is the user side is not defined.

The ideal approach is to create a symmetric call establishment algorithm that can unambiguously select each call's bearer channel(s) without collisions or negotiations. Such a protocol does exist if we treat the two directions of the connection independently. The key is that call collision exists because a single bearer channel number is used to specify both directions of the call. If we negotiate each direction separately some powerful protocol simplifications can occur. We call this approach "two-way independent channel selection".

The two-way independent channel selection concept is quite simple. Each end of an ISDN link can be made master over the transmit direction (relative to itself). To establish a call, the calling party sends the called party a Setup message containing the Bearer Service information element, a Channel Identification information element that specifies the bearer channel(s) for the calling-to-called direction of the call, and any optional information elements required. When the called party receives the Setup message it decides whether or not it can accept the call. If it can, the called party will then send a Connect message (optionally preceded by an Alerting or Call Proceeding message) to the source. To insure compatibility, only the Connect message contains the Channel Identification information element for the called-to-calling direction of the call.

Notice that no changes to I.451 need be made to create this scheme. A few simple restrictions to the call set up options insure true compatibility across all implementations. One point that should be discussed is the impact two-way independent channel selection will have on the implementation of the communication equipment. There are two things to consider: how difficult is two-way independent channel selection to implement, and what is lost by not permitting bearer channel negotiation.

First the implementation. For discussion purposes let us use the communication equipment model of Figure 3-5.

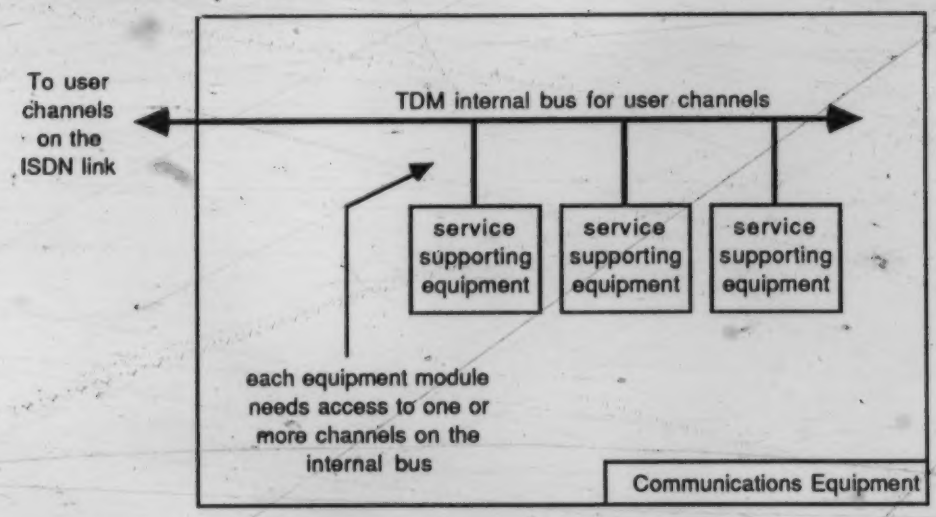

To user channels on the ISDN link
TDM internal bus for user channels
service supporting equipment
service supporting equipment
service supporting equipment
each equipment module needs access to one or more channels on the internal bus
Communications Equipment

Fig. 3-5. Communication Equipment Model

The main difficulty with two-way independent channel selection is that there can be no fixed mapping between the channels of the equipment's internal bus and the channels of the ISDN link. However, there are numerous methods available to insure the equipment has dynamic access to any channel on the interface. One method is to have each equipment module dynamically pick off the user channels needed. The signalling processor can inform each module which channel(s) it should be looking at. Another method is to place a Time Slot Interchange (TSI) switching component between the link and the internal bus. The signalling processor controls the TSI such that the time slots (channels) coming in from the link are dynamically switched into an appropriate channel on the internal bus. In this way, each equipment module can read a fixed channel location on the internal bus regardless of the actual interface channel the data arrives on. Both methods are fairly standard, and add little complexity to the implementation.

Next we need to decide whether the loss of channel negotiation capability will adversely effect the capabilities or implementation of the communications equipment.

The question, of course, is why support channel negotiation at all? Probably the most important reason is to overcome the call collision problem discussed in the last chapter, but two-way independent channel selection eliminates this problem. A secondary issue is that the equipment may want to prevent the use of certain channels on its internal bus because they have failed or are busy. If there is a fixed correspondence between channels coming in on the communication link and channels on the internal bus then the equipment may want to control which incoming channels are available for use. Another way to overcome this is by decoupling the communications link from the internal bus using a TSI switch. As long as the number of channels in the TDM link, the TSI switch, and the internal bus match there is no difficulty in mapping any channel in the link to any channel in the internal bus. This type of setup is shown (for the receive direction only) in Figure 3-6. Note that the transmit direction does not require a TSI between the link and the internal bus since the equipment is master in the transmit direction, although one could be added for flexibility.

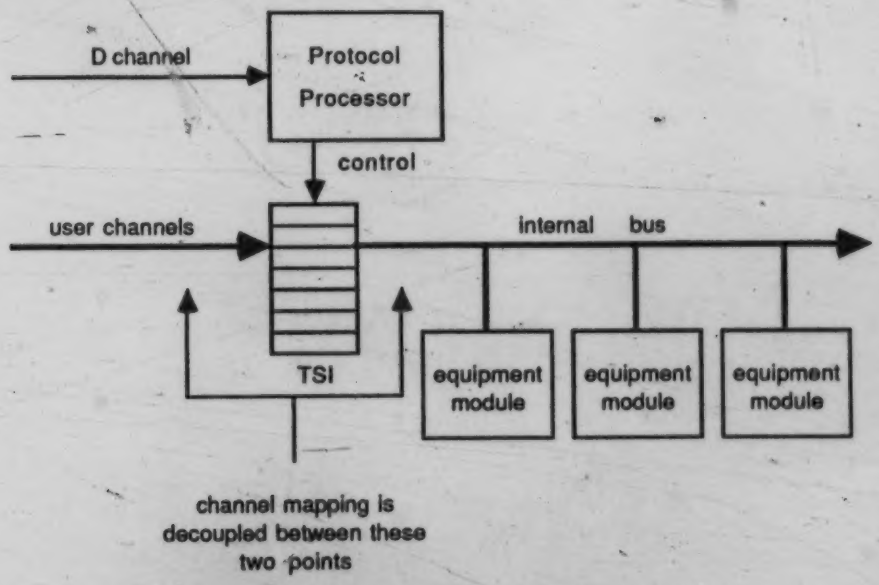


Fig. 3-6. ISDN Interface with a Time-Slot Interchange

The complete subset protocol is specified at the end of this chapter, but to summarize the salient features of the proposed call establishment procedure:

1) Stimulus terminals are not supported.
2) Bearer channel negotiations are eliminated using two-way independent channel selection in which each end of the link is master over its transmit direction.

3) There is less freedom concerning the information elements embedded in the various messages. Specifically, the Setup message and Connect message must contain the Channel Identification information element, and it is not permitted in any other message.

4) Note that nothing in points 2 or 3 effects the type of services that can be established using the streamlined protocol. Complexity is reduced by placing greater restrictions on the implementation options, not on the service options of the network layer.

Point 4 is very important since it implies that applying restrictions properly creates a fully featured but simplified call set up protocol. As will be shown next, this philosophy applies equally well to the other stages of the call.

3.5.3. Active Call Management

Once a call enters the active state, the users can communicate directly via the established bearer service. Normally an active call requires no additional D channel signalling until it is time to take the call down and release the bearer channels (user channels) that are supporting the bearer service. However, there are situations in which it is useful to be able to modify the call's parameters in some way without requiring the call to be taken down and set up again. A simple example of this is call transfer, but there are many others. In general, any changes to an active call's parameters will require additional signalling on the D channel, and hence the network layer will have to support these call modification requests during the active state of the call. Unfortunately, this area of ISDN signalling was left for further study in the Red Book version of I.451.

Although much work has been done by the CCITT in this area since 1984, the protocol selected for our simulation does not support modification of an active call other than call disestablishment. The reason for this is straight forward: it is an area that is in a state of change, and adding additional messaging capabilities to the network layer's active state is not difficult to do when needed. The ability to modify a call while it is in progress is an important feature to support for services such as switched voice, but it has been left for future work to implement these capabilities as they become better defined by the CCITT.

A second area of call rearrangement specified by the CCITT is the ability to suspend and resume a call on the 2B+D basic rate interface. Temporary call suspension permits terminals to be relocated or even replaced by another terminal while the call is still active. The analogy is moving today's phone from one jack to another or substituting a

phone with a modem. The ability to suspend a call is certainly useful, but like call modification it is something that can be added to the subset by later work as the need arises. Our simulated protocol does not support call suspension.

So far we have stated that although two areas of ISDN signalling during an active call state have not been implemented they can be added easily. As a partial proof of this the subset protocol does support one type of active call signalling: user-to-user information exchange. Basically, user-to-user signalling is the exchange of arbitrary information between the two end users. In this case the D channel simply provides the information routing and delivery service much like a packet network. There is no attempt by the network layer to look at or interpret the information being sent. User-to-user signalling during call set up and take down is supported in I.451 by permitting the inclusion of an "User-user Information" information element in most of the messages used during these stages of the call control procedures. But what about while the call is in the active state?

Our protocol defines a new internal request to initiate the transmission of a User Information message in which is embedded a User-user Information information element. It is a simple addition which provides an example of how signalling during the call's active state is handled.

3.5.4. Call Disestablishment

The procedure for disconnecting an established connection is not as complex as for setting it up, but still there are some issues that must be addressed. There are essentially two options for call disestablishment. The difference between the two pivots on the importance of the call reference to the link equipment, as described below.

There are two resources that are uniquely assigned to each call by the link equipment. The first is the bearer channel on the link, which was the focus of the call establishment discussion . The second is the call reference, which is a unique 8 or 16 bit identification number assigned to every call at set up time. From a protocol point of view, the call reference is simply the tag used to associate each signalling message to the call it is controlling. From the implementation point of view, however, the call reference is extremely important since it will be used as a "pointer" into the equipment's internal information tables. These information tables store critical call information such as channel assignment, billing information, and call state.

The reason the call reference is being discussed here is that controlling the assignment and release of call references is critical to maintaining proper control of the link. When a call is taken down, the bearer channels can be freed and re-assigned

immediately. Not so the call reference. The call reference must remain unavailable for reassignment until both ends have completed all post-call processing such as recording connect time for billing. If one end releases then immediately reassigns a call reference before the other end has released it, messages intended to set up the second call may corrupt the internal status of the first call. At best, the second call's Setup message will be rejected by the destination because of call state conflicts.

The simplest form of I.451's call disestablishment procedure is designed to insure both sides have truly released the call's assigned resources before they are freed for re-assignment. To this end, the normal call take down procedure involves a three way handshake. To initiate disestablishment one end sends a Disconnect message with the call reference assigned to the service to be disestablished. Upon receipt of the Disconnect message the other end releases the bearer channel(s) assigned to the call and replies with a Release message. The end that originated the disconnect then responses by releasing both the channel(s) and the call reference and replies with a Release Complete message, which permits the other end to finally release its call reference.

But the Red Book version of I.451 defined a variation on the above procedure by allowing a Detach message to be sent instead of the Release message in response to a Disconnect. The idea is that the bearer channels can be released, but the call reference must be retained until a later Release or Disconnect is sent. Why this would actually be used and by what type of equipment is not defined. Strictly speaking the ability to detach is not required to take down a call. If the Disconnect, Release, Release Complete procedure was the only one available it may reduce an implementation option, but it need not effect the type of services available. Therefore, the I.451 subset used for our simulations did not support the ability to detach. As it turns out, this whole discussion may be academic because it appears that detach may be eliminated from the next release of the CCITT ISDN standards in the 1988 Blue Book. The August 1987 version does not mention detach procedures.

One further point is worth mentioning. The most recent version of I.451 defines a Restart procedure which permits one side to reset the other side, where reset means that all calls are immediately cleared. This is used to recover from a major failure. Because they are still fairly new and somewhat complicated the subset protocol does not support these procedures.

3.6. The Protocol Subset

In this section the I.451 protocol used in our simulation is defined and justified. The justification really must answer two questions: "Why is what is in the subset protocol

there?", and "Why is what is in the full I.451 protocol not there?". The section opens with a presentation of the protocol processor model used by the rest of the section to explain the various aspects of the subset protocol. It may seem odd to open a discussion of what should be an implementation independent protocol specification by presenting an implementation model, but the reality is that the need for "states" and "internal requests" makes much more sense when described using this model. The model is not part of the I.451 recommendation although it does shed some light on the nature of the original I.451 recommendation. Although the model was developed while exploring the implementation of the signalling system's protocol processor the model is not meant to restrict the implementation of the protocol. For now it is simply a tool used in its definition. The implementation details of the protocol processor have been left for the next chapter.

The remaining subsections describe and justify the subset protocol. Much of the discussion will focus on the SDL diagram found in Appendix A. This diagram combined with the call control procedures outlined in this section define our simplified signalling protocol. The section closes with an analysis of how to model both the user and network sides of the I.451 protocol using an extension to the signalling system model.

3.6.1. Protocol Processor Model

In the ISDN signalling system model presented at the beginning of this chapter a functional block called the "signalling protocol processor" is shown. In this subsection we present a further refinement of this block in order to create a protocol processor model to be used for subsequent discussions. The first level of partitioning is to separate the physical, data link and network layers in accordance with the OSI reference model. The next step is to partition the network layer processing into two modules: a Protocol Specific Module (PSM) and an Application Specific Module (ASM) as is discussed below.

The discussions of the previous section focused on the handshaking procedures that are followed by the I.451 network layer protocol. The protocol defines when each type of message can be sent and what type of information elements are mandatory or optional in each message. The exchange of messages in an appropriate handshaking procedure insures the orderly exchange of signalling information between the ends of the link, but it is the information elements themselves that actually convey the detailed signalling information. Further, although each application is required to follow the message exchange protocols (as defined by the SDL diagram), different classes of applications may actually require different information elements be exchanged. Some of the optional information elements may actually be mandatory for a particular implementation in

order to insure compatibility. As well, some optional information elements may not be needed for certain applications, so why generate them if they do not contain information for which meaningful processing has been defined? Addressing information is a good example. I.451 has defined an information element called Calling Party Number that can be used to pass the telephone number of the calling party to the called party. In this way the customer will be able to determine who is calling before picking up the phone. From the operating company's point of view this capability is somewhat controversial, so not all countries or operating companies will support it. An operating company may dictate that ISDN switching equipment installed in its domain must have the Calling Party Number information element removed from all messages.

To further complicate matters, not all equipment will process the signalling information in the same way. For example, assume a user's multi-media terminal is connected to a central switch (i.e. the standard telephony configuration). If the terminal receives a Setup message it must check to see if the service being requested is available, and then configure its internal bus so that the bearer channel on the link is connected to the equipment that supports that service. The addressing elements of the Setup message can be ignored by the terminal if the switch ensures that only those messages that are addressed to the terminal are actually sent to the terminal. When the switch receives a Setup message from a terminal, it must check the destination address element to determine where the call should be connected to. The switch then looks at the Bearer Capability element and the Channel Identification element in order to determine how to create the requested connection. The switch must notify the called party (assume its another terminal on the same switch) of the incoming call by sending it a Setup message. Any user-to-user information contained in the original Setup message must be routed to the Setup message being sent to the called party terminal. Obviously, much of the hardware necessary to implement the network layer in a terminal could be very different from that required by a switch.

But the above example demonstrates an important point: it is possible to separate those aspects of the processing that can be shared across applications such as the initial processing of the messages (decoding, checking the structure of the message, etc.) from those aspects that are strictly dependent on the application and its inherent hardware and/or software structure. The concept is actually quite simple. The protocol specific aspects of the network layer insure that the call procedures are followed and that only valid information is exchanged between the ends of the link. The application specific aspects insure the signalling information is actually processed according the the type of

application being implemented. This separation of duties leads to the partitioning of the network layer into two modules, the PSM and the ASM as show in Figure 3-7.

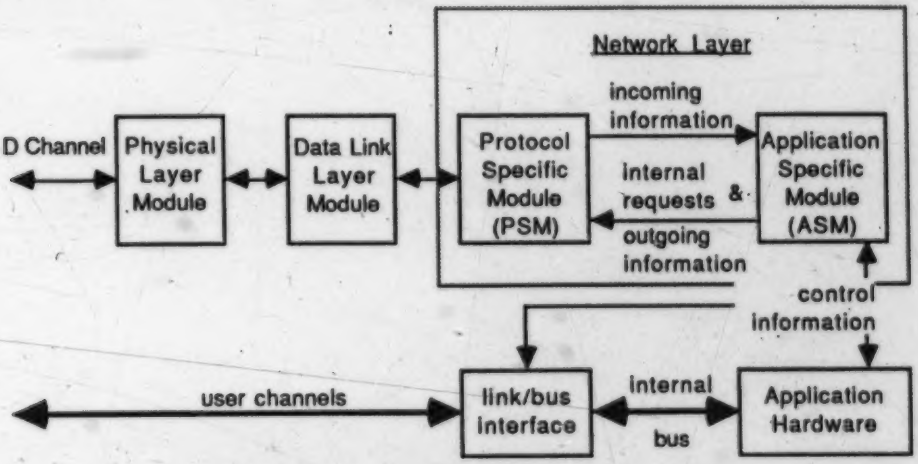


Fig. 3-7. The Protocol Processor Model

A few notes on the model are in order:

1) The term "application" is used to refer to the type of ISDN communication equipment being implemented. The application hardware is mainly the user channel processing equipment. Its nature depends on the application being implemented. It is this module that is ultimately the source of the service control commands since it is through this module that the user interfaces to the communications equipment. If the application is a switch or PBX then the application hardware is the switching fabric plus the interface to the control system that is managing the establishment and disestablishment of the service connections.

2) The combination of PSM and ASM create the ISDN signalling system's network layer protocol processor. The PSM and ASM are embedded in the communication equipment to control the user channel services defined by the application. The PSM is essentially a "front-end" for the ASM. The PSM handles the details of the handshaking required by the signalling protocol being implemented while the ASM is the module that actually executes the actions indicated by the signalling.

3) The PSM's chief responsibility is to maintain each call's state machine (as defined by the SDL diagram). Calls are identified (addressed) by their unique call reference. The PSM checks each incoming message to insure it is valid to receive such a message given the state of the call to which the message is addressed. Further, it checks the type of information elements embedded in the message to

insure they are valid within the message. The procedures followed when errors are detected are discussed below.

4) The PSM will immediately respond to a received message by transmitting a reply message only if the reply requires no information content that is application specific. Specifically, the PSM can only autonomously generate messages in response to error conditions and when the reply is a simple acknowledgement message.

5) The PSM passes the information embedded in the incoming message to the ASM for processing. Although the PSM eliminates illegal message and information element types, it does not actually check the contents of the information element. Information element processing is left up to the ASM.

6) The PSM accepts internal requests from the ASM and updates the state of the appropriate call reference. If the internal request results in a message being sent out, the PSM can only supply the message type and call reference. The ASM must supply all other information elements. This is because the contents of the information elements are application specific, and the PSM by definition is application independent.

7) The ASM controls the application's user channels as indicated by the signalling information received from the application hardware and from the other end of the link via the D channel signalling. The exact nature of the ASM differs widely depending on the application being implemented. Although each application's ASM may need to perform different internal control actions to configure the communication equipment in response to signalling events, the details of each type of ASM can be hidden from the PSM. The PSM is only concerned with maintaining the overall signalling protocol.

8) The SDL diagram insures a call state is defined at each point in the procedures where the PSM must wait for information from the other side (in the form of a message) or from the ASM (in the form of an internal request). No intermediate state is required when the PSM receives a message and generates the response automatically. Therefore, the protocol processor model is useful in understanding why the original states in the I.451 protocol appear where they do.

In conclusion, the protocol processor model is extremely useful in explaining the nature of our I.451 subset. The differentiation between the ASM and PSM becomes clearer in the following subsections.

3.6.2. Error Handling Procedures

The purpose of this subsection is to explain the general error handling procedures of the protocol subset. The procedures are called general because they apply regardless of call state. This greatly simplifies the SDL diagrams by eliminating the need to explicitly cover every possible event in every possible state.

It seems that no matter how robust the physical and data link layers are and no matter how much handshaking is done in the network layer it is still impossible to guarantee that something somewhere will not cause the two ends of the link to become out of step during the signalling procedures. Because of this it is necessary to provide the network layer with some form of protocol timeout facility. This facility permits the network layer to set a timer while waiting for an event to occur. If the timer goes off before any of the appropriate events occur then the network layer assumes something has gone wrong somewhere and begins recovery actions.

The timeout facility selected for our subset protocol is a straightforward simplification of the more complex timer specifications of the full I.451 protocol. A single timer with a single reset value is required for each call reference. When the call is in the Null, Active, or Alerting states the timer is stopped. In all other states the timer must be running. If a timeout occurs it is treated exactly like an internal clear request, or in other words, actions are taken to initiate call disestablishment. Once these actions have been initiated they cannot be stopped; the call must be taken down. Unlike the full I.451, there are no "retry on timeout" facilities. A timeout is effectively a non-recoverable failure of the call's signalling protocol so the call must be terminated.

The SDL diagrams in Appendix A do not explicitly show timeout handling procedures since the timeout procedure for a given state is identical to the procedure for an internal clear (call take down) request except the cause field of the Cause information element of the Disconnect or Release message used to initiate the call disestablishment is set to cause 111 - "undefined protocol error". By defining the timeout procedures in this way, we have minimized the complexity added by the timeout facilities and insured that a timeout will clear the call in an orderly manner.

With regard to timeout values, it is felt that a single relatively long timeout period is all that is needed for the simplified signalling protocol. The reason for this is quite clear; timeouts are simply not expected to happen very often. This will indeed be the case if three criteria are met by the implementation:

1) Network layer messages very rarely get through the data link layer errored (or lost).

2) The signalling system (including the D channel) will rarely fail. This not only includes hardware failure but also "soft" failures such as protocol processing errors causing inappropriate actions to be taken.

3) The protocol processor has as one of its requirements the ability to handle, without loss of information, any level of valid signalling activity.

The first two points are self-explanatory, but the third may appear somewhat strong at first glance. Does it not imply that the protocol processor must have infinite processing capability? The answer, of course, is no. First of all, the D channel has a finite bandwidth capability so incoming messages can only arrive so fast. Secondly, as long as internal requests are logically "debounced" and are queued without loss they can be processed at a finite rate. This ability should be standard with any good design so making it a requirement is nothing drastic. And finally, the protocol itself creates a fully locked handshaking that naturally dampens the amount of legitimate signalling that can occur. This precludes the scenario in which the far end suddenly bursts out hundreds of messages and swamps the message handling system. It does not preclude the far end from "going insane" and sending a continuous stream of structurally valid but meaningless messages, but this is not a valid signalling activity so it is excluded from point 3.

In summary, the timeout facility specified for the subset protocol is robust yet simple. Experimental results from the protocol simulation have proven the timeout facility will take down calls that experience an errored or missing signalling message(s) during call establishment or disestablishment. Since the timeout procedures mimic the normal call clearing procedures this can also be shown directly by tracing the appropriate paths in the SDL diagram.

The handling of error conditions was not well defined in the Red Book I.451, but in the August 1987 version of I.451, section 5.8, the error handling procedures have been carefully laid out. Generally one of three procedures is followed in response to receiving a message that is in error:

1) Ignore the message and do not change state.

2) Ignore the errored information element but process the rest of the message.

3) Ignore the message and initiate clearing of the call.

In procedures 1 and 2 there may or may not be a Status message sent out in response to the error. Status messages are used by I.451 to return error information to the other end of the link. The Status message contains four types of information: the call reference, a cause, a diagnostic, and a call state. The call reference value is the same as the call reference of the received message that has been found in error. The contents of the cause field and its associated diagnostic field are defined by I.451. The call state is the current

state of the call corresponding to the call reference of the errored message. The cases when Status messages occur are described below.

The I.451 protocol defined for the simulation stayed as close as possible to the standard error condition handling procedures. Small differences occur when it was felt a "special case" which would be difficult to implement was defined unnecessarily, or when the error procedure conflicted with the PSM/ASM partitioning used for our implementation. Clearly the latter is more serious than the former should complete compliance with I.451 become the goal, but for this stage of the research it was felt the trade-offs were justified.

Before discussing error conditions for received messages it is probably timely to show how a normal I.451 message is formed:

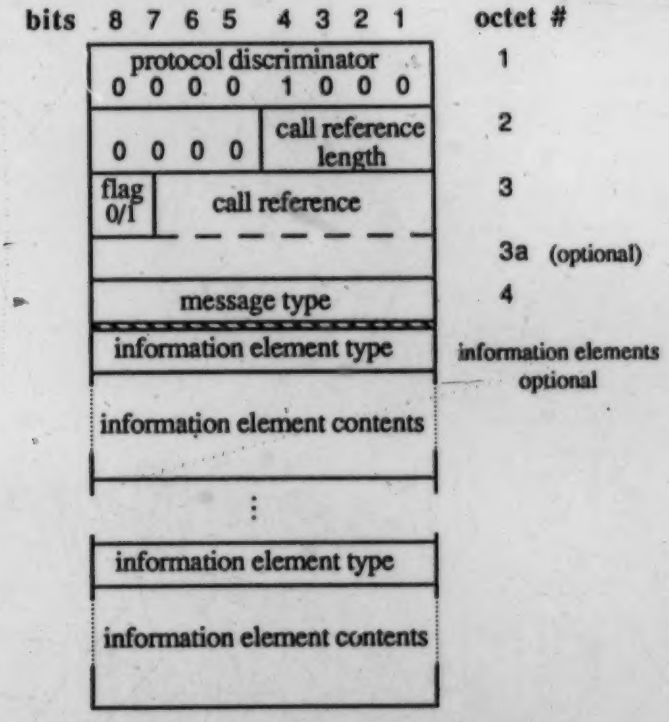


Fig. 3-8. I.451 Message Structure

The following procedures define how error conditions are handled. Note that the normal procedures for handling received Disconnect, Release, Release Complete, and Status messages is defined by the SDL diagrams in Appendix A.

1) Poorly formed messages are ignored, where ignored means do nothing, as if the message was not received. Poorly formed messages are those with one or more of the following properties:

- A protocol discriminator information element other than 00001000.
- A length of call reference octet where bits 8-5 are other than 0000.
- A call reference length value greater than what is supported by the equipment.
- A total length less than the minimum required to contain the protocol discriminator, the call reference length, the call reference, and the message type.

2) If a message other than Status is received with a call reference of 0 and the flag bit either 0 or 1 then a Status message is returned with cause value 81 - "invalid call reference". The diagnostic is the call reference. A Status message with a call reference of 0 is considered valid, but the PSM simply passes it to the ASM.

3) Whenever a message is received with a message type that is not implemented (or illegal) a Status message is sent out with a cause value of 97 - "message type non-existent or not implemented". The diagnostic is the message type. The message is not processed and no state change occurs.

4) When a known message is received in any state other than the null state (state 0) that is not expected for that state, a Status message is sent out with a cause value of 101 - "message not compatible with call state". The diagnostic is the message type. The message is not processed and no state change occurs.

5) When an unexpected message is received in the null state (state 0) a Release Complete message is sent out with a cause value of 81 - "invalid call reference". The message is not processed and the call remains in the null state.

6) If a Status message is received for a call reference that is in the null state, but the Call State information element contains a call state value other than Null (0) then a Release Complete message shall be sent with cause value 101 - "message not compatible with call state". The diagnostic is the message type.

7) If an otherwise valid message is received with one or more mandatory elements missing a Status message is sent out for each missing information element with a cause value of 96 - "mandatory information element missing". The diagnostic is the missing information element type. The message is not processed and no state change occurs. The exceptions to this rule are the Disconnect, Release and Release Complete messages where the release cause is not included. In these cases the disestablishment procedures are continued as normal but the reply Release or Release Complete message will use a cause value of 96. In terms of implementation, the easiest way is to treat the Cause information element as optional with the default return Cause being 96.

8) If a message is received with any information elements repeated, the PSM passes all elements to the ASM for processing. The ASM must decide how to handle the duplication. If the message is Release and the Cause information element is repeated it is the last cause that is used in the reply Release Complete message. Information element ordering is not checked by the PSM.

9) The Setup message has its own set of rules which override the above rules. If it is received in any state other than null it is ignored. If its call reference has the flag bit set to 1 (called side) it is ignored. If any mandatory information elements are missing then a Release Complete message is returned with cause value 96.

The only difference in the error handling procedures of the full I.451 protocol and the subset is how information element content errors are handled (section 5.8.6.2 of the August 1987 I.451 recommendation). The difficulty is that this error is not supposed to result in a state change: the message or just the errored information element should be ignored depending on whether it is a mandatory or optional information element, respectively. By our definition of the PSM/ASM partitioning the PSM does not check information element contents, and hence it will pass the errored message to the ASM and change states as if the message was valid. The ASM will have to respond an errored mandatory information element by taking down the call with a Clear Request. Differences of this type are probably not worth worrying about since errored messages should be rare if the data link layer is doing its job.

3.6.3. Call Establishment

The call establishment phase is certainly the area in which the most significant simplifications were achieved by the subset protocol. If we compare the "stock" SDL diagram taken from the August 1987 I.451 (Appendix B) the reduction in complexity becomes obvious. The use of Alerting and Call Proceeding messages has been simplified and several states in the original SDL diagram have been removed due to the rationalization of timeout procedures, the elimination of stimulus terminals, and the lack of bearer channel negotiations due to the adoption of the two-way independent channel selection technique.

The Alerting message can be interpreted as notification that the called side is alerting the user and hence the protocol timer at the calling side should be stopped, while the Call Proceeding message is simply a way for the called side to let the calling side know that it is still alive but needs more time to process the call. The differences between call establishment in the full I.451 and our subset are delineated and justified below:

1) State 2 is unnecessary if stimulus terminals are not supported and channel negotiations are not required. The Setup message can and should contain all the information necessary (except for the called party to calling party bearer channel identification of course).

2) State 1 can be collapsed into state 3 since there is no difference between the two when using our timer definition. The protocol simply resets the timer and remains in state 3 when a Call Proceeding message is received.

3) State 4 is collapsed into state 3 since receiving the Alerting message can result in the timer being stopped without the need to change states. Obviously if a Call Proceeding message is received after an Alerting message the timer will be restarted, but that might be a nice feature to have anyway.

4) Simply to eliminate the doubt, the Connect Acknowledge option is eliminated: Connect messages are always replied to with a Connect Acknowledge.

5) At the called side states 6, 7, & 25 can be collapsed into state 9. The reasoning parallels points 1, 2, & 3 above. Notice that a Setup request is rejected by the ASM via a Clear Request which results in a Release Complete message being sent. This is somewhat non-standard if an Alerting or Call Proceeding message has already been sent since a Disconnect message should be used in these cases. However, just using Release Complete will work even when interfacing to a full I.451 implementation, it is the least complex approach, and it is consistent with our PSM/ASM partitioning and the proposed state reductions.

Viewed "from the outside", these call establishment procedures are not radically different than stock I.451; the ability to respond to a Setup message with a Call Proceeding, Alerting and Connect message has been retained. The addition of the Delay Request internal event is necessary to clarify when a Call Proceeding message will be sent.

So far we have been discussing the handshaking required to establish a call at the message level. What about the actual contents of the messages being sent and received? Or put another way, what restrictions should be placed on the information elements of each type of message in order to minimize complexity? Since I.451 fully defines which information elements are permitted to appear in each of the various messages we need only discuss those instances in which the subset protocol differs from the full protocol. However, it is worth noting that the large number of optional information elements defined for almost every message type certainly puts a burden on any implementation. In a well behaved application such as an ISDN LAN/PBX it would be advantageous to greatly reduce the number of information element types to process.

As has already been discussed, the two-way independent channel selection procedure requires the Channel Identification information element to be mandatory in the Setup and Connect messages, and illegal in the Alerting, Call Proceeding and Connect Acknowledge messages. This differs slightly from the standard I.451 where Channel Identification was optional for all five messages except for the network-to-user direction in the Setup message where it was mandatory. The other difference is in the interpretation of the Channel Identification information element since in the subset it only applies to a single direction. Fortunately this can be handled naturally by the existing coding of the Channel Identification element.

The protocol processor model implies that all application specific information must, by definition, be generated by the ASM. Therefore it is important to insure that any messages sent automatically by PSM message are not permitted to contain information elements that would have to come from the ASM. What this means in terms of the SDL diagram is that any message that is sent out by the PSM in direct response to a received message (as opposed to an internal request generated by the ASM) must only contain information the PSM can derive from the received message. Messages sent out by the PSM in response to an internal request do not have this restriction because the request can include those information elements the ASM wishes the PSM to tack on to the message. This implies that since the PSM generates Connect Acknowledge automatically the Connect Acknowledge message cannot contain any information elements other than the minimum set: protocol discriminator, call reference, and message type. Therefore, in the subset protocol no optional information elements are permitted in the Connect Acknowledge message. The tables and figures in Appendix A summarize the differences in message content between the full and subset protocols. The other messages sent by the PSM are discussed in the next subsection.

3.6.4. Call Disestablishment

The subset's call disestablishment procedures are similar to the equivalent I.451 procedures for symmetric implementations except that the subset does not support call suspension. Since states 3 and 9 have had other states "collapsed" into them the action to be taken for an internal Clear Request or timeout had to be selected carefully. Call disestablishment initiated from the Outgoing Call Proceeding and Connect Request states is handled with a Disconnect message. This is because clearing a call in these states is somewhat abnormal and hence the full call take down procedure is desirable. This is consistent with I.451 procedures. Call disestablishment from state 9 was already

discussed in the previous section. The rest of the call disestablishment procedures are identical to the new symmetric version of I.451.

It is worth repeating at this point that the SDL diagrams in Appendix A do not explicitly show a "Timer Expired" internal event. This is because,in order to reduce complexity, timeouts are assumed to be handled internally by the PSM as if a Clear Request had occurred. The definition of Clear Request events in states 11 and 19 may seem a bit odd but it provides the procedure to be followed for a timer expired event. Defined in this way it also permits the ASM to implement the timer functions instead of the PSM if this becomes desirable.

Some comments on the information elements of the Disconnect, Release, and Release Complete messages are in order. I.451 defines the Cause information element as mandatory for all three messages. This appears to break our rule that the PSM cannot generate application related information elements. The Release Complete messages is always generated autonomously by the PSM, and the Release message is generated in response to Disconnect in the Disconnect Request state. The way around this is to enforce a rule that when the Cause information element must be generated by the PSM the Cause is simply an echo of the Cause sent in the original Disconnect or Release message. In this way no ASM intervention is required. It does add some additional complexity to the PSM since the Cause information element from incoming Disconnect and Release messages must be saved temporarily, but this is not too difficult to implement. Of course the Release Complete message (and the Release message in some cases) may not contain any other information elements for the same reason as the Connect Acknowledge message.

3.6.5. User Information Requests

As was mentioned in the previous section, the ability to support user information requests was added to the subset protocol as an example of how signalling activity during the Active state of a call would be handled. To this end, a new internal request was defined such that the ASM could notify the PSM when a User Information message should be sent. From the SDL diagram it is obvious that handling user information requests in the active state is quite simple. If there is user information to be added to messages in other states then the User-user Information information element can be added to the outgoing message by the ASM just like any of the other optional information elements.

3.6.6. Extending the System Model

In the system model section it was mentioned that it is desirable to be able to extend our signalling system model to cover a communications network in which only the ISDN signalling protocols are used. Since the physical and data link layers are by definition

concerned only with a single transmission link it is the network layer protocol which must be extended. The trick is to define the network side protocol using two network layer protocol processors running back-to-back. The linkage between the two network layer processors appears as information exchanged internally by the switch. This type of structure can be modelled as shown in Figure 3-9.

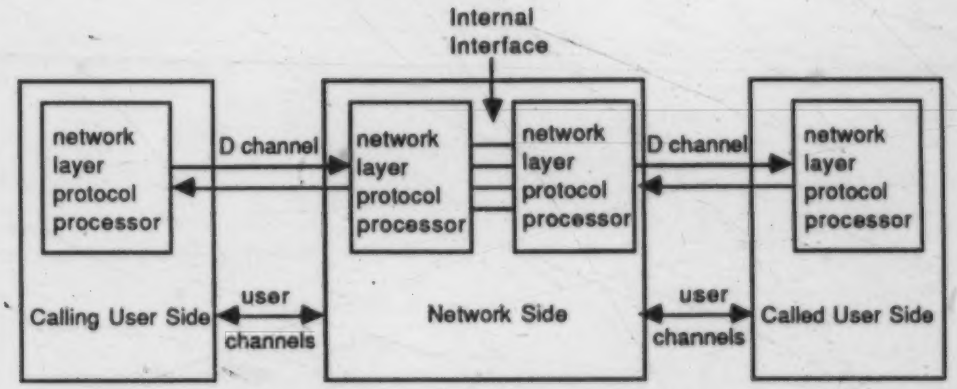


Fig. 3-9. Extended Network-Side Model

In the general case, any number of intermediate switching stages can be required to complete a terminal-to-terminal connection. This situation is modelled by connecting each of the individual links in the back-to-back manner shown in Figure 3-10. The rule used by each switch to translate a message received by the protocol processor on one side of the switch to the internal request passed to the protocol processor on the other side of the switch is very simple: generate the same internal request type that caused the received message to be generated at the node that sent it. The effect of this rule becomes obvious if you run through the call establishment or disestablishment procedures using the sample connection shown below: messages generated by the calling or called terminal propagate across the connection in an end-to-end fashion.

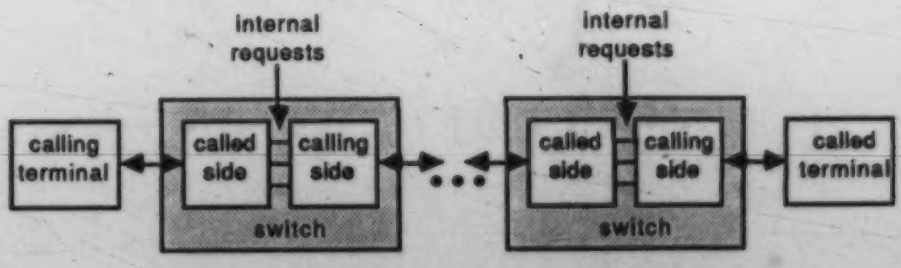


Fig. 3-10. Multiple Switch Connection Model

Chapter 4

Protocol Processor Design

4.1. Chapter Overview

The previous chapter analyzed the overall functionality required of the physical, data link, and network layers of a B-ISDN signalling system. This chapter extends that discussion with a more rigorous specification of the required functionality and with a detailed description of the interfaces necessary between the various modules of the protocol processor.

These interfaces have been defined with hardware implementation in mind, but the module functionality is defined using pseudo-code algorithms as opposed to hardware circuits. This is consistent with the level to which the research was taken. The algorithms and interfaces are somewhat slanted to how the software simulation was designed, but the algorithms are not the suggested implementation, they are simply the functional description of the module. Obviously when hardware implementation is attempted there will be changes made in the way things are done. That is to be expected, but in general the algorithms and interfaces described here should be adequate to define the type of functionality and partitioning one would expect to see in the hardware implementation.

Because we will be discussing at length the implementation of the various logical modules that make up our protocol processor model it is convenient to introduce a bit of nomenclature. The abbreviations L1, L2, and L3 will be used to denote the implementation of the physical layer (the D channel), the data link layer, and the network layer modules, respectively. Further, since the protocol processor model accurately represents the partitioning used for our implementation the terms PSM and ASM will also be used to mean the implementation of the logical PSM and ASM defined in the model. Finally, the term "protocol processor" should now be taken to mean the implementation of the protocol processor model. For example, the phrase "the PSM and ASM are sub-modules of L3 which is a module of the protocol processor" is well formed. The primary partitioning of our implementation appears in Figure 4-1.

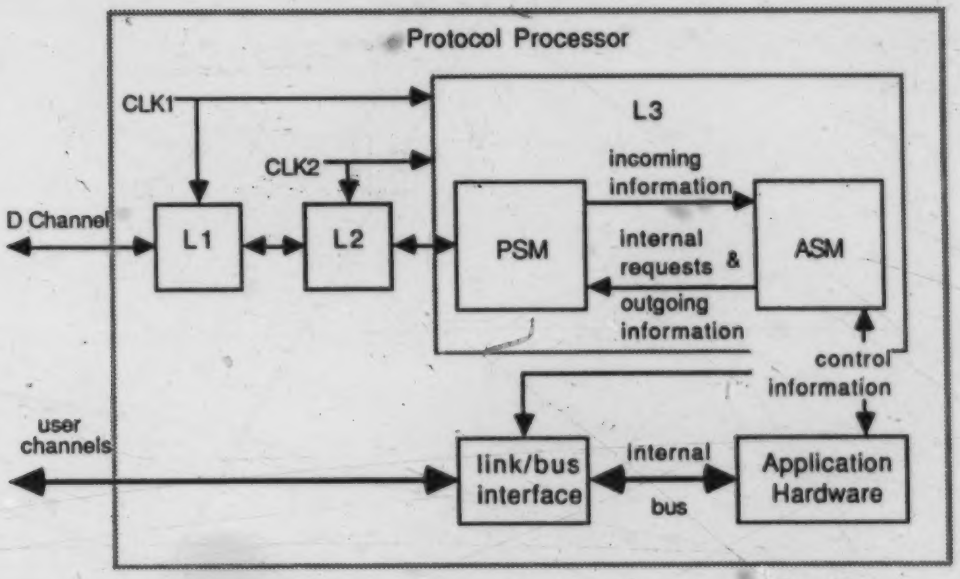


Fig. 4-1. Partitioning of the Protocol Processor Implementation

The physical and data link layers do not receive a lot of attention in this chapter. The actual implementation of the physical interface of an ISDN link is not a key element in the research since the software simulates the D channel using a simple RS-232 serial connection between two computers with a fully locked handshaking protocol to control the timing synchronization between the two ends of the link. Thus the D channel used for the simulation has little in common with a standard ISDN D channel. Since the functionality required of L1 was well covered in the last chapter, this chapter concentrates more on the interface between L1 and L2 than on the way L1 would actually be implemented in a B-ISDN link. The algorithm used by the simplified B-ISDN data link layer is described in detail in this chapter, as is the interface between L2 and L3. There was no need to further partition L2 since its algorithm is straightforward.

It is the network layer that receives most of the attention in this chapter. Because the ASM is, by definition, unique to each application, the discussion of the ASM's functional requirements focuses mainly on its interface to the PSM. Only general observations can be made concerning the "application hardware" aspects of the ASM. It was not intended that the initial research go much beyond the PSM/ASM interface since a realistic specification for the ASM cannot be made without first restricting the application to a specific piece of ISDN equipment within a specific ISDN network environment. This level of detail was not compatible with the time and resource constraints of a MSc program.

The chapter opens by discussing the very important issue of timing and synchronization. In the next section the interfaces between L1, L2, and L3 are defined. Here the concept of "stream processing" is introduced. The next three sections define the functionality expected of L1, L2, and L3 respectively. L3's PSM is further partitioned in order to better structure the definition of the PSM's functional requirements and to better explain the PSM/ASM interface. In the next chapter the software simulation of the protocol processor is discussed.

4.2. Timing Considerations

In this section the overall timing and synchronization of the protocol processor is described. As ISDN is currently defined, the user side of an ISDN interface derives its transmit clock from the receive side of the transmission link. Therefore the user side is "loop-timed" to the network side. In a symmetric configuration there will have to be a synchronization plan established among the communication equipment such that overall synchronization is maintained. In general it is probably safe to say that the transmit and receive directions of the transmission link will be synchronized, but this whole synchronization issue is a complex aspect of telecommunication network design and it is well beyond the scope of this thesis to go into it any deeper. Fortunately we need not be concerned with synchronization beyond analyzing how the protocol processor should interface to the incoming and outgoing D channels.

Difficulties with the implementation of systems with more than one clock source will occur at the points in the system where signals cross a synchronization (clock source) boundary. It is known that care must be taken in the implementation of these interfaces or non-deterministic behavior could result.[15] If we assume that the protocol processor's system clock is independent from the transmission link clock then these boundaries occur not only at the interface between the protocol processor and the D channel, but also between the protocol processor and any application equipment that is processing user channels and is synchronized to the user channels' clock. Clearly the "narrow view" would suggest it is best to have the protocol processor's system clock derived from the transmission link's clock, but unless all components of the communication equipment's control system derive their timing from this single clock source there will still be synchronization boundaries to contend with. If the equipment is an ISDN interface to a computer, for example, it may not be possible to avoid having the CPU and the ISDN transmission system clocked from different sources.

So where does this all lead? Since it is not possible to give a definitive answer to the system clocking problem, the best we can do is to ensure that the following criteria are met:

1) Any clocks used within the protocol processor are synchronized with each other.
2) If the protocol processor must interface to modules that are not clocked from the same system clock as the protocol processor then appropriate design techniques will be incorporated into the interface to compensate for this such that the protocol processor design is not effected.

Given the level of implementation to which the research has gone these constraints are sufficient to permit us to ignore the synchronization issue. However, there is still the issue of timing among the various modules within the protocol processor. In the partitioning diagram shown above there are two clock signals (CLK1 and CLK2) shown as inputs to the L1, L2, and L3 modules. Two clocks are used in the design to facilitate a "pipeline" architecture in which a signal (or information if you prefer) is processed in successive stages. In the protocol processor the incoming D channel is processed first by L1, then by L2, and finally by L3. Outgoing messages follow the reverse path, originating in L3 and exiting through L1. Adjacent stages of the pipeline are clocked by non-overlapping clocks in order to facilitate simple signal timing and handshaking between modules. This technique was used mainly to simplify the simulation software since timing based on a single clock has some undesirable properties from the interface simulation point of view. For example, assume signals can only be read on the rising edge of the clock and must be modified before the trailing edge. In Figure 4-2, a "command-response" type of information exchange between modules using a one clock scheme is shown. The two phase clock scheme appears in Figure 4-3.

The advantage of the two phase clock scheme is that the A node need not "skip a cycle" before it can look at the response line. It is the need to implement this wait state which complicates the implementation of the A node. Two phase clocking permits the A node to initiate the command in one cycle and process the response in the next with no need for an intermediate wait-state. There are certainly numerous clocking schemes that could and should be entertained for the actual hardware implementation of the protocol processor, but the two phase clock technique is well suited to the needs of the simulation.

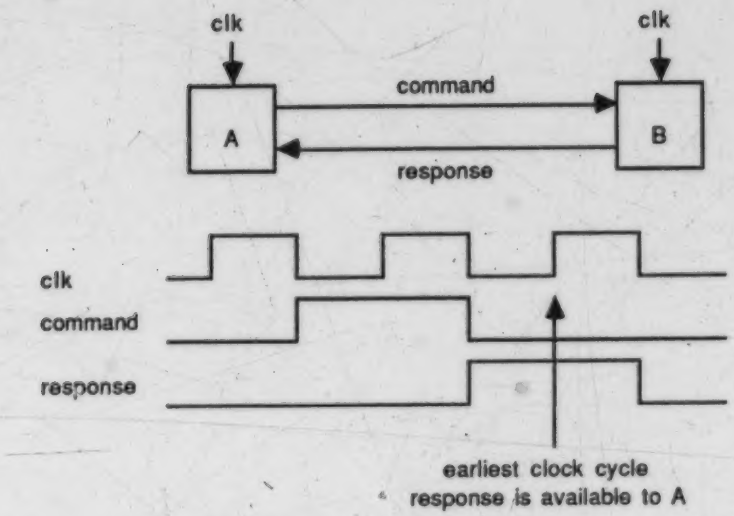


Fig. 4-2. Single Clock Handshaking

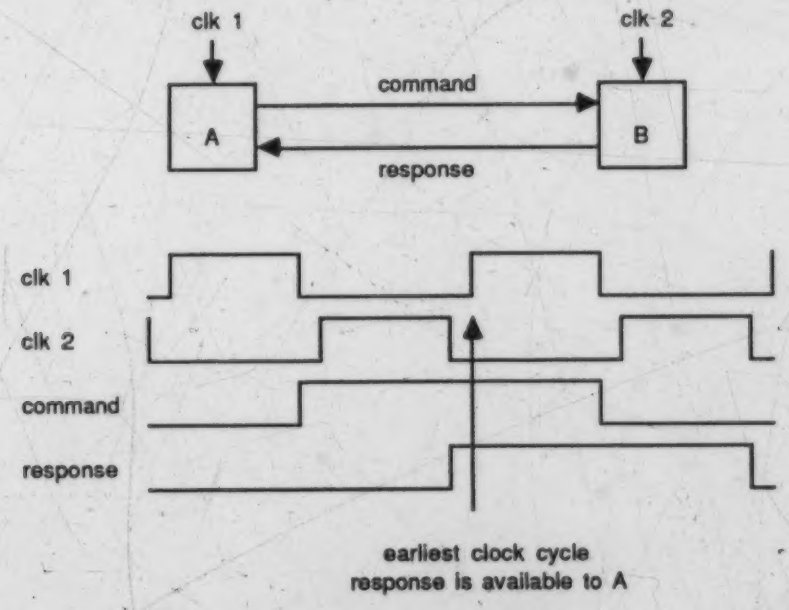


Fig. 4-3. Two Clock Handshaking

4.3. Interfaces Between Major Modules

In the previous chapter there was considerable discussion concerning the use of forward error correction (FEC) on the B-ISDN D channel and how that can simplify the data link layer and influence the data link layer to network layer interface. Some key features that warrant restating are:

1) The physical layer (D channel) uses additional bandwidth to mark each byte on the D channel as either no data (null), start of message (som), data internal to the message structure (msg), or end of message (eom). Further there may be extra check bits added on a per byte basis (used for FEC) that may override the primary classification and define the byte as errored (error).

2) The data link layer must check the structural integrity of the packets it receives. This implies every packet must be of the form <som> <msg> <msg> .... <msg> <eom>. If this structure is not found, or if a <null> or <error> byte occurs between a <som> and <eom> then the packet is classified as errored. Optionally, the last <msg> byte and the <eom> byte could be a 16 bit CRC although this was not the case in our implementation.

3) There is no need for the data link layer to store up the packet's bytes as they come in and then pass the complete packet to the network layer once its integrity has been confirmed. Layer 2 can pass layer 3 the bytes as they arrive so that processing can begin on the signalling message as soon as it starts to arrive. The only catch is that layer 3 has to be able to "undo" any of its actions should layer 2 subsequently classify the message as errored.

Given these features, the definition of the L1-L2 and L2-L3 interfaces is straightforward. Graphically they appear as in Figure 4-4. The various signals are defined in Table 4-1.

In our simulation the D channel exchanges a new byte of data every four CLK1 clock periods. In general, if the transmission link is B-ISDN then the D channel interface to L1 need not be a consistent multiple of CLK1 clock periods. This is because the D channel must be inserted into the overall 125 μsecond frame of the transmission link, and there will be several D channel bytes inserted into every frame if the D channel's data rate is greater than 64 Kb/s. The average D channel bit rate will be constant, but the rate at which the D channel's bytes arrive at L1 may jitter slightly.

Therefore, L1 notifies L2 when new D channel data has arrived and when the next D channel byte can be sent out. The latter situation is complicated somewhat by the fact that the outgoing D channel must come through L2 from L3. Since L2 adds one clock

period delay to the D channel (in both directions) L1 uses the L1-L2 DATA BUS LOCK signal to "warn" L2 that the next CLK1 clock period will be the one in which the outgoing D channel is latched by L1 and sent out. L2 uses the L1-L2 DATA BUS LOCK signal to initiate the latching of the outgoing D channel from L3 and the updating of the L2-L1 DATA bus and L2-L1 DATA TYPE signals. This ensures that on the next CLK1 clock cycle L1 is sending the most up-to-date D channel value from L3. Timing diagrams of typical data flow are shown in Figures 4-5 & 4-6.

Notice that the data and data type signals between L1&L2 and L2&L3 in the L1->L2->L3 direction are marked as x and x' (x={a,b,c,d}) respectively. That is because L2 processing may effect the value of the L1-L2 signals. However this is not the case in the L3->L2->L1 direction since L2 does no real processing of the data or data type in this direction. L2 processing is defined in detail in a later section, but it is important to emphasize that our L2 processing is on a byte-by-byte basis. L2 interfaces with L3 at the byte level, not at the packet level as is defined for LAPD. If L3 cannot immediately process the incoming D channel data it must buffer it temporarily within the L3 module. This type of byte-by-byte interface is what we call "stream processing" since the D channel flows through the protocol processor pipeline as a continuous stream.

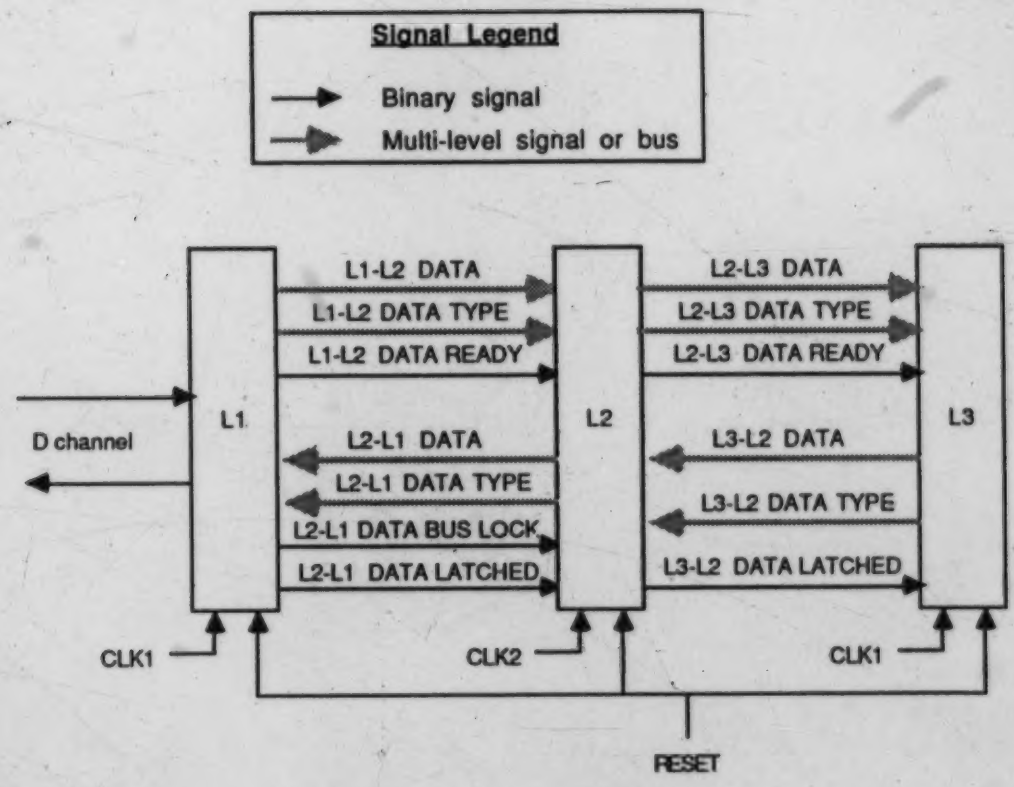


Fig. 4-4. L1-L2-L3 Interfaces

Signal Name	Signal Type	Signal Definition
CLK1	clock line	non-overlapping two-phase clocking
CLK2	clock line	non-overlapping two-phase clocking
RESET	binary signal	when asserted causes reset of internal states of L1, L2, & L3
L1-L2 DATA	8 bit wide data bus	D channel data from L1 to L2
L1-L2 DATA TYPE	3 bit, 5-level	Classifies L1-L2 DATA as one of <null>, <som>, <msg>, <eom>, <error>. Others values are undefined.
L1-L2 DATA READY	binary signal	Remains asserted for one CLK1 clock period when L1-L2 DATA contains new data.
L2-L1 DATA	8 bit wide data bus	D channel data from L2 to L1
L2-L1 DATA TYPE	2 bit, 4-level	Classifies L2-L1 DATA as one of <null>, <som>, <msg>, <eom>.
L2-L1 DATA BUS LOCK	binary signal	Is asserted for the CLK1 clock period immediately preceding the clock period in which the data on L2-L1 DATA BUS is latched from L2 by L1 and sent out on the D channel.
L2-L1 DATA LATCHED	binary signal	Is asserted for the CLK1 clock period in which the data on L2-L1 DATA BUS is latched from L2 by L1 and sent out on the D channel.
L2-L3 DATA	8 bit wide data bus	D channel data from L2 to L3
L2-L3 DATA TYPE	2 bit, 4-level	Classifies L2-L3 DATA as one of <som>, <msg>, <eom>, <error>.
L2-L3 DATA READY	binary signal	Remains asserted for one CLK2 clock period when L2-L3 DATA contains new valid data.
L3-L2 DATA	8 bit wide data bus	D channel data from L3 to L2
L3-L2 DATA TYPE	2 bit, 4-level	Classifies L3-L2 DATA as one of <null>, <som>, <msg>, <eom>.
L3-L2 DATA LATCHED	binary signal	Is asserted for the CLK2 clock period in which the data on L3-L2 DATA BUS is latched from L3 by L2. The data will be latched from L2 by L1 on the next CLK1 cycle and sent out on the D channel. This signal is enabled by the L1-L2 DATA BUS LOCK signal.

Table 4-1. L1-L2-L3 Signals Definition

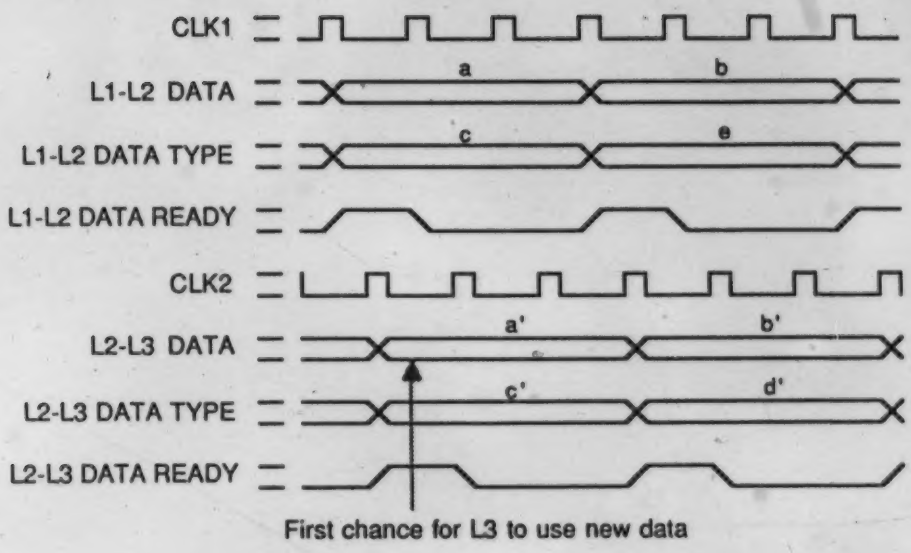


Fig. 4-5. L1->L2->L3 Data Flow Timing

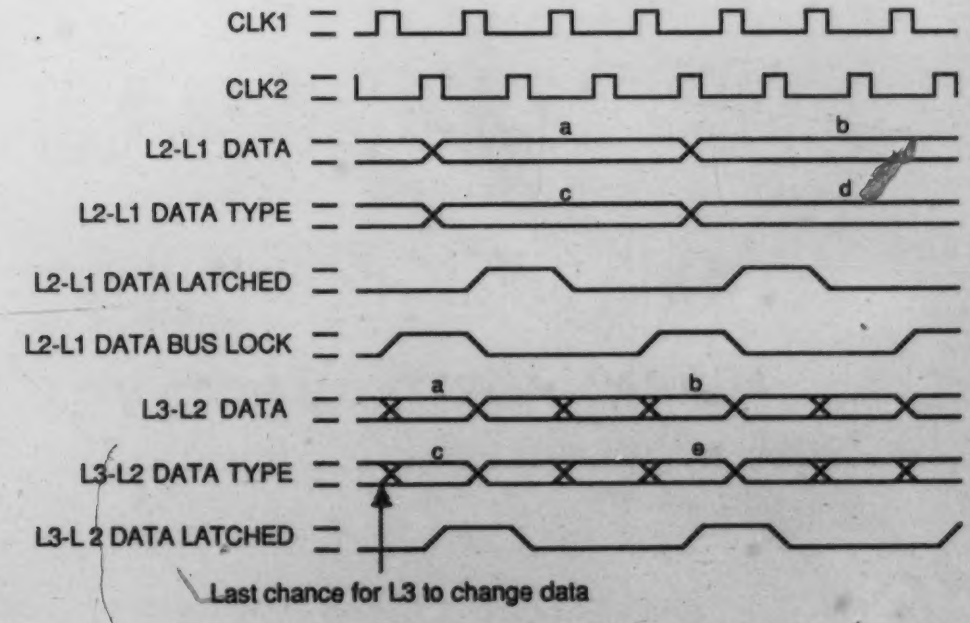


Fig. 4-6. L3->L2->L1 Data Flow Timing

4.4. Physical Layer Implementation

The functionality expected of the signalling system's physical layer is well covered in the previous chapter. With respect to hardware implementation, the L1 module used for the protocol processor simulation does not represent the implementation of an actual B-ISDN D channel, so there is little that can be added in this section. In the next chapter the technique used to simulate D channel timing and the L1 interface are discussed.

4.5. Data Link Layer Implementation

As was discussed in the previous chapter, by enhancing the functionality of the D channel the implementation of the data link layer can be greatly simplified. On the transmit side (with respect to the D channel) of the L2 module the processing to be done is extremely simple. In fact, in the L3 to L1 direction the data link layer is simply a "null layer". The algorithm used by L2 in this direction is as follows:

```
if power up or RESET then
    L2-L3 DATA READY := <deasserted>
else when CLK2 goes high do
    if L1-L2 DATA BUS LOCK is <asserted> then
        L2-L1 DATA            := L3-L2 DATA
        L2-L1 DATA TYPE       := L3-L2 DATA TYPE
        L3-L2 DATA LATCHED    := <asserted>
    else
        L2-L1 DATA            := no change
        L2-L1 DATA TYPE       := no change
        L3-L2 DATA LATCHED    := <deasserted>
```

Notice that our L2 does not actually use the L2-L1 DATA LATCHED signal. If for hardware design reasons it was simpler to assert the L2-L1 DATA bus for only one clock period and then let it float (or change as L3-L2 DATA changes) then the CLK2 clock cycle in which L2-L1 DATA LATCHED is asserted is the earliest clock period in which L3-L2 DATA and L3-L2 DATA TYPE could be changed after L1 reads them.

In the receive direction (with respect to the transmission link) the L2 algorithm ensures that packet's structural integrity is maintained. The algorithm is presented below. Notice that L2 ensures that every L2-L2 DATA TYPE sequence consists of a SOM, followed by any number (including 0) <msg>, and terminated with either a <eom> or <error>. No L1-L2 DATA TYPE = <null> are allowed within the sequence, and the <null> D channel values are not passed on to L3 since L2-L3 DATA READY is only asserted when the data type is <som>, <msg>, <eom>, or <error>.

```
if power up or RESET then
    set internal state register: SkipToSOM := <true>
    L2-L3 DATA READY := <deasserted>
else when CLK2 goes high do
    if L1-L2 DATA READY is <asserted> then
        {
        if SkipToSOM is <true> then
            if L1-L2 DATA TYPE is <som> then
                SkipToSOM           := <false>
                L2-L3 DATA TYPE     := <som>
                L2-L2 DATA          := L1-L2 DATA
                L2-L3 DATA READY    := <asserted>
            else
                L2-L3 DATA TYPE     := no change
                L2-L2 DATA          := no change
                L2-L3 DATA READY    := <deasserted>
        else
            L2-L3 DATA READY := <asserted>
            case L1-L2 DATA TYPE of
                <som>  : L2-L3 DATA TYPE    := <error>
                         L2-L3 DATA         := <don't care>
                         SkipToSOM          := <true>
                <msg>  : L2-L3 DATA TYPE    := <msg>
                         L2-L3 DATA         := L1-L2 DATA
                         SkipToSOM          := <false>
                <eom>  : L2-L3 DATA TYPE    := <eom>
                         L2-L3 DATA         := L1-L2 DATA
                         SkipToSOM          := <true>
                <error>: L2-L3 DATA TYPE    := <error>
                         L2-L3 DATA         := <don't care>
                         SkipToSOM          := <true>
        }
    else
        L2-L3 DATA READY := <deasserted>
```

4.6. Network Layer Implementation

The first step in describing how L3 is implemented is to further partition the PSM module into three sub-modules: the RxFifo, the TxFifo, and the Event Processing Module (EPM). The two FIFOs are used to temporarily buffer the incoming and outgoing D channel data streams, while the EPM is the heart of the PSM; it implements the PSM functions defined in the previous chapter and controls the PSM/ASM interface.

The section begins with a discussion of the FIFO buffering technique used. The partitioning of the PSM is described and justified. The RxFifo-EPM interface and the TxFifo-EPM interface are then defined. In the next two subsections the algorithms used to implement the RxFifo and the TxFifo are described. This is followed by the EPM subsection in which implementation of the PSM functions is described. A detailed PSM/ASM interface specification can be found in this subsection as well. The section closes with a brief look at the hardware implementation aspects of the ASM.

4.6.1. FIFO Based Message Flow Control

At first look, the need for message buffering may seem inconsistent with our previously stated philosophy of "stream processing" . The problem is that there are actually three sources of events that the protocol processor must handle: incoming message events, internal request events, and the protocol timeout event. Each of these can cause information messages to be sent out, thereby creating the need to allocate the outgoing D-channel to one message source at a time, while buffering the others. Further, while timeouts and internal requests are being processed, it is necessary to buffer the incoming D channel until the EPM has finished processing the other event.

There are several methods that could be used to solve the buffering problem. In order to select among the alternatives, it is first necessary to examine the situation more closely:

1) Due to the "state machine" nature of the I.451 protocol, decisions as to which event source (message, request, or timeout) to process can only occur at a stable state. Therefore, once the processing of an event has begun it should be processed to completion and the next stable state should be entered before the processing of a new event is started. However, this applies only to events related to a single call reference. Events corresponding to different call references could be processed in parallel except when it comes time to access the outgoing D-channel, which is an inherently serial resource.

2) The critical processing occurs while processing an internal event (request or timeout) and an incoming message starts to arrive. Depending on the message there may or may not be an immediate outgoing message in response, where immediate means the next stable state is entered only after the reply message is sent. When an immediate response is required, the transmission of any outgoing message related to the previously processed internal event must be completed before sending the response message. Fortunately, when an immediate response is required it is always a very short message of known length since no ASM generated information elements are allowed.

3) Only incoming messages can "overflow" the protocol processor. Requests and timeouts can be handled one at a time, with the ASM responsible for insuring that requests and timeouts are queued while waiting for the PSM. It is also true, however, that local events must not be "locked out" for an excessive period of time if timeouts at the far end are to be avoided.

4) If ASM generated information elements are permitted in a message its length cannot be predetermined easily. The type and length of information elements included in the message can vary in length depending on the type of service being established.

5) During the processing of an internal request, timeout, or message that requires immediate response, there is a point at which the processing is effectively complete (including the assignment of the next state) but yet the outgoing message generated in response to the event has not yet been completely sent. Before this point is reached, only one event can be processed using serial hardware, but after this point in the processing there exists an opportunity for "pipelining". If this is exploited properly it could be effective in optimizing throughput with little or no impact on buffer requirements, and without the need to include redundant hardware for parallel processing.

The buffering solution selected in response to these constraints is shown in Figure 4-7. The various signals are defined in Table 4-2. The partitioning and signal definitions are explained fully in the following subsections. The functionality of the two FIFOs is defined first, although their importance to the overall implementation of the PSM is not explained until the EPM's functionality is defined and the EPM/ASM interface is fully specified.

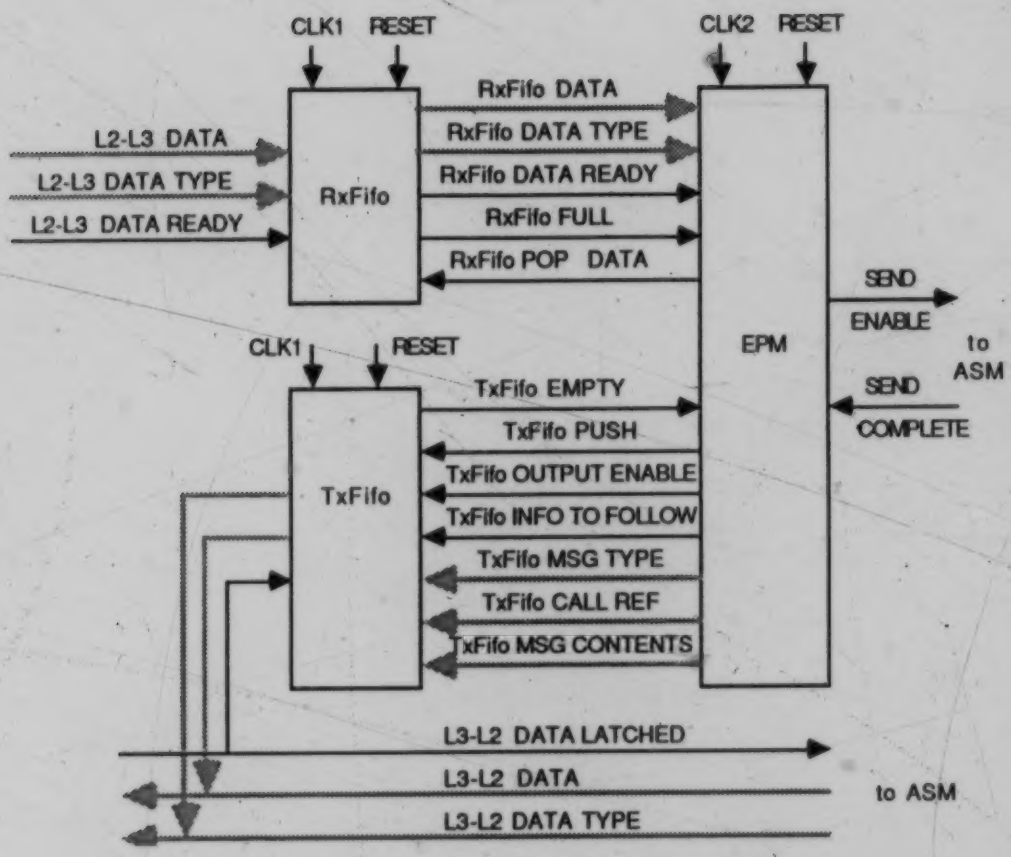


Fig. 4-7. PSM Partitioning into Sub-Modules

Signal Name	Signal Type	Signal Definition
CLK1	clock line	non-overlapping two-phase clocking
CLK2	clock line	non-overlapping two-phase clocking
RESET	binary signal	When asserted causes reset of RxFifo, TxFifo & EPM
RxFifo DATA	8 bit wide data bus	D channel data from L2
RxFifo DATA TYPE	2 bit, 4-level	Classifies RxFifo DATA as one of <som>, <msg>, <eom>, <error>.
RxFifo FULL	binary signal	Asserted if FIFO is storing more than a preset number of D channel data and type bytes.
RxFifo DATA READY	binary signal	Asserted if RxFifo DATA & RxFifo DATA TYPE are new values. When RxFifo DATA READY is deasserted then the data is undefined.
RxFifo POP DATA	binary signal	On the CLK1 clock cycle following the assertion of RxFifo POP DATA the next value of RxFifo DATA & RxFifo DATA TYPE are popped off the FIFO.
TxFifo EMPTY	binary signal	Asserted if TxFifo is empty.
TxFifo PUSH	binary signal	Asserted by the EPM to get the TxFifo to push a message onto the FIFO.
TxFifo OUTPUT ENABLE	binary signal	Asserted if TxFifo is permitted to place data on the L3-L2 DATA bus. L3-L2 DATA LATCH acts as the POP signal to the TxFifo.
TxFifo INFO TO FOLLOW		binary signal Asserted if the ASM will be appending information elements to the message initiated by the TxFifo. This signal determines whether the last data byte the TxFifo sends out should be an <eom> or a <msg> type.
TxFifo MSG TYPE	n bit, m-level	Message type TxFifo is to send out.
TxFifo CALL REF	n bit, m-level	Message's call reference value.
TxFifo MSG CONTENTS	multiple busses	Passes the information element contents created by the EPM to the TxFifo.
SEND ENABLE	binary signal	Asserted to inform ASM that it can begin to append information elements to the outgoing message initiated by the EPM/TxFifo. This signal is part of the overall EPM-ASM interface defined later in the thesis.
SEND COMPLETE	binary signal	Asserted to inform EEM that ASM is finished appending information elements to the outgoing message. This signal is part of the overall EPM-ASM interface defined later in the thesis.
TxFifo DATA LATCHED	binary signal	Is asserted for the CLK1 clock period in which the data on TxFifo DATA BUS is latched from L2 by L1 and sent out on the D channel.
external signals		See interface section above.

Table 4-2. PSM Internal Signals Definition

4.6.2. RxFifo Sub-module

The RxFifo is used to buffer the incoming D channel data byte and data type in a standard First In, First Out (FIFO) data store. It can store up to N (data byte, data type) pairs, with a special value M < N defined as the "getting close to full" point. There are two operations defined for the FIFO, push and pop. The push operation causes the current values of L2-L3 DATA and L2-L3 DATA TYPE to be pushed into the tail of the FIFO buffer. Push is triggered on the rising CLK1 edge when L2-L3 DATA READY is asserted. If the FIFO is not empty the Pop operation takes the head of the FIFO and pops the pair into the RxFifo DATA and RxFifo DATA TYPE output busses. Pop is triggered on the rising CLK1 edge when RxFifo POP DATA is asserted. The RxFifo DATA READY signal is only updated when a pop operation has been performed or when it was deasserted in the previous clock cycle but on this clock cycle a new value has been pushed into the FIFO.

FIFO overflow should never occur if N and M are selected correctly as is explained in Section 4.6.4.2. However to be safe, an overflow procedure has been defined. The trick is to truncate the last packet from L2 that will not fit into the FIFO with an <error> data type and then throw away all subsequent packets until at least one pop operation is performed and the data type from L2 is <som>. The RxFifo can use the fact that L2 ensures all packets end with either a <eom> or <error> to simplify this procedure. The complete algorithm for the RxFifo implementation is as follows:

```
if power up or RESET do
    RxFifo FULL  := <deasserted>
    RxFifo DATA READY := <deasserted>
    initialize the FIFO to empty
    RxFifo DATA & RxFifo DATA TYPE are undefined
    set internal state register: OverflowDetected := <false>
else on CLK1 going high do
    {
    (check for push operation first)
    if L2-L3 DATA READY is <asserted> then
        if OverflowDetected is <true> then
            if L2-L3 DATA TYPE is <eom> or <error> then
                OverflowDetected := <false>
        else
            if FIFO is full then
                OverflowDetected := <true>
                if data type at tail is not <eom> then
                    change data type at tail to <error>
            else
                push L2-L3 DATA and L2-L3 DATA TYPE onto the FIFO
```

```
(check for pop operation)
if FIFO is empty then
    if RxFifo POP DATA is <asserted> then
        RxFifo DATA & RxFifo DATA TYPE are undefined
        RxFifo DATA READY := <deasserted>
    else
        no changes
else  (fifo is not empty)
    if not RxFifo DATA READY or RxFifo POP DATA then  (if was empty or popped then)
        pop head of fifo into RxFifo DATA & RxFifo DATA TYPE
    RxFifo DATA READY := <asserted>
if there are greater than M elements in the FIFO then
    RxFifo FULL := <asserted>
else
    RxFifo FULL := <deasserted>
}
```

4.6.3. *TxFifo Sub-module*

The TxFifo is actually much more than a simple FIFO. It is also a complete message generator for the EPM and the message initiator for those messages that the ASM adds information elements to. Every TxFifo PUSH request results in the TxFifo adding the "standard message header" - a protocol discriminator octet (00001000), a call reference length octet (00000001 or 00000010), the TxFifo CALL REF (1 or 2 octets depending on the standard), and the appropriate message type (1 octet). Table 4-3 defines how the TxFifo handles the request for each type of message that is permitted in our subset I.451 protocol. The standard message header is always pushed onto the FIFO with the protocol discriminator having a type of <som> and the length and call reference octets having a type of of <msg>. For some messages the message type octet will be given a <eom> type, for others a <msg> type. For simplicity, the message type octet is abbreviated MTO. Refer to Appendix A for a set of diagrams showing the basic contents of the subset protocol's valid messages.

The TxFifo differentiates between EPM and ASM generated Disconnect and Release messages by the value of the TxFifo INFO TO FOLLOW signal. If it is asserted then the ASM will be supplying the information elements, otherwise the EPM supplies them. The TxFifo gets the EPM generated information element values from the TxFifo MSG CONTENTS bus. The structure and timing of this bus need not be defined here. It is sufficient to say that if the bus is not wide enough to accommodate a one shot parallel transfer of the worse case information content (cause, diagnostic and call state) then a multiple clock cycle transfer will have to be used.

Message Type	Element Source	Procedure Definition

Alerting	ASM	if TxFifo INFO TO FOLLOW is <asserted> then
		MTO type is <msg>
		else
		MTO type is <eom>

Call Proceeding	ASM	if TxFifo INFO TO FOLLOW is <asserted> then
		MTO type is <msg>
		else
		MTO type is <eom>

Connect	ASM	The ASM must add the Channel Identification information element so MTO is always <msg>
Connect Ack	-	There can be no information elements to add, so MTO type is always <eom>.
Disconnect	ASM	The ASM must add the Cause information element so MTO is always <msg>.
	EPM	The EPM must supply the cause value for the Cause information element. No diagnostic field is required. The MTO type is <msg>, followed by the Cause information element (see Appendix A). The last octet of the Cause information element is set to <eom>.
Release	ASM	The ASM must add the Cause information element so MTO is always <msg>.
	EPM	The EPM must supply the cause value for the Cause information element. No diagnostic field is required. The MTO type is <msg>, followed by the Cause information element (see Appendix A). The last octet of the Cause information element is set to <eom>.
Release Complete	EPM	The EPM must supply the cause value for the Cause information element. No diagnostic field is required. The MTO type is <msg>, followed by the Cause information element (see Appendix A). The last octet of the Cause information element is set to <eom>.
Status	EPM	The EPM must supply the cause & diagnostic values for the Cause information element and the call state value for the Call State information element. The MTO type is <msg>, followed by the Cause information element and then the Call State information element (see Appendix A). The last octet of the Call State information element is set to <eom>.
User Information	ASM	The ASM must add the User-user information element so MTO is always <msg>.

Table 4-3. TxFifo Message Handling Definition

Table 4-3 defines what the push operation of TxFifo looks like, so now all that is needed is the overall definition of the FIFO processing. Notice that the algorithm does the pop before the push. This actually could be the other way around as well since the pop (L3-L2 DATA LATCH) will occur whether the FIFO is empty or not. The TxFifo algorithm is defined as follows:

```
if power up or RESET do
    TxFifo EMPTY := <asserted>
    initialize the FIFO to empty
    L3-L2 DATA := 0
    L2-L2 DATA TYPE := <null>
else on CLK1 going high do
    {
    if TxFifo OUTPUT ENABLED is <asserted> then
        if L3-L2 DATA LATCHED is <asserted> then
            if FIFO is not empty then
                L3-L2 DATA      := data head of the FIFO
                L3-L2 DATA TYPE := data type head of the FIFO
                pop and throw away the head of the FIFO
            else
                L3-L2 DATA := 0
                L3-L2 DATA TYPE := <null>
        else
            L3-L2 DATA & L3-L2 DATA TYPE are not changed
    else
        L3-L2 DATA & L3-L2 DATA TYPE are tri-stated
    if TxFifo PUSH is <asserted> then
        if there is not enough room in the FIFO for the message then
            ignore the push command
        else
            push the appropriate message onto TxFifo's tail
    if the FIFO is empty then
        TxFifo EMPTY := <asserted>
    else
        TxFifo EMPTY := <deasserted>
    }
```

4.6.4. EPM Sub-module

In the previous chapter the general requirements of the PSM were discussed and defined, as was the separation of PSM from ASM. In this subsection the key sub-module of the PSM - the EPM - is specified. In the previous three subsections the FIFO based interface between L2 and the EPM has been defined, but it remains to be explained exactly how the EPM uses these buffering facilities to implement the PSM's functionality. But first it is necessary to fully define the interface between the EPM and the ASM. This is followed by the specification of the event queuing technique used by the EPM. Next the implementation of the call reference data structures is analyzed. The subsection then closes with a full description of the EPM's algorithm.

4.6.4.1. The EPM-ASM Interface

The EPM-ASM interface defined here is the implementation of the protocol processor model's PSM-ASM logical interface. Like the L2-PSM interface, the EPM-ASM interface is oriented to a stream processing approach. Once the EPM has stripped off the protocol discriminator and the call reference from the incoming message it simply

passes the message type and any subsequent information elements to the ASM on a byte-by-byte basis. At any point after the message type is passed but before the final end-of-message indication is passed the EPM can suddenly notify the ASM that an error in the message has been detected. The ASM must then "undo" whatever it may have done with the previous contents of the message. Further, the EPM does not check the integrity of the information element contents as was discussed in the previous chapter.

This stream processing approach is evident in the signal definitions of the EPM-ASM interface shown in Figure 4-8 and defined in Table 4-4.

Most of the EPM-ASM signals are used to pass D channel or timeout information to the ASM. The role of these signals will be defined more fully when the EPM algorithm is specified. The other key signals are used to control the processing of the ASM's internal event requests. These signals are REQUEST CONTROL, EVENT REQUEST STATUS, AND INFO TO FOLLOW. The two call reference signals identify which call the other signals apply to. There are several scenarios for how the handshaking between the EPM and the ASM can proceed when the ASM wishes to initiate an <alerting>, <clear>, <connect>, <call proceeding> or <user information> internal event request. These are shown in Figures 4-9 to 4-12. Note that if a signal is not specified explicitly in the figure then its default value is assumed. The default values are defined in the signal legend of Figure 4-9.

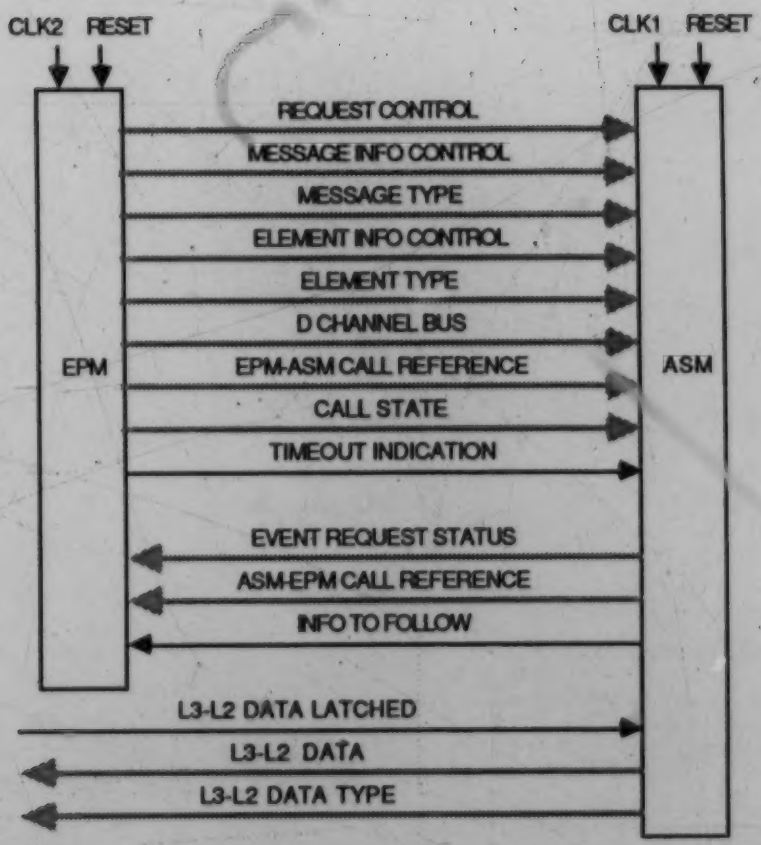


Fig. 4-8. The EPM-ASM Interface

Signal Name	Signal Type	Signal Definition
CLK1	clock line	non-overlapping two-phase clocking
CLK2	clock line	non-overlapping two-phase clocking
RESET	binary signal	When asserted causes reset of ASM & EPM
REQUEST CONTROL	3 bit - 5 level	<none>, <request ack>, <request reject>, <request complete>, <clear to send>. Unused levels are undefined.
MESSAGE INFO CONTROL	3 bit - 5 level	<none>, <som>, <msg>, <msg good>, <msg bad>. Unused levels are undefined.
MESSAGE TYPE	4 bit - 10 level	<alert>, <call proceeding>, <connect>, <connect acknowledge>, <disconnect>, <release>, <release complete>, <setup>, <status>, <user information>. Unused levels are undefined.
ELEMENT INFO CONTROL	3 bit - 5 level	<none>, <som>, <elm>, <elm good>, <elm bad>. Unused levels are undefined.
ELEMENT TYPE	n bit - m level	<bearer capability>, <cause>, <call state>, <channel identification>, <user-user information> + <optional information elements defined for the protocol>. Unused levels are undefined.
D CHANNEL BUS	8 bit bus	Used to pass the information element contents from the EPM to the ASM.
EPM-ASM CALL REFERENCE	8 or 16 bit bus	Used to pass the call reference value from the EPM to the ASM.
CALL STATE	3 bit - 8 level	<null>, <outgoing call proceeding>, <incoming call proceeding>, <connect request>, <active>, <disconnect request>, <disconnect indication>, <release request>.
TIMOUT INDICATION	binary signal	Asserted if a protocol timeout has occurred and the call specified by EPM-ASM CALL REFERENCE is being taken down.
EVENT REQUEST STATUS	3 bit - 8 level	<no request>, <request pending>, <request cleared>, <alert request>, <clear request>, <connect request>, <delay request>, <user information request>.
ASM-EPM CALL REFERENCE	8 or 16 bit bus	Used to pass the call reference value from the ASM to the EPM.
INFO TO FOLLOW	binary signal	When asserted with an alert or delay request it tells the EPM that the ASM wants to append some information elements to the Alerting or Call Proceeding message that will be sent out by the TxFifo. For other internal requests there are mandatory information elements that must be added by the ASM, so the signal is assumed to be asserted.
external signals		Defined previously.

Table 4-4. EPM-ASM Interface Signals Definition

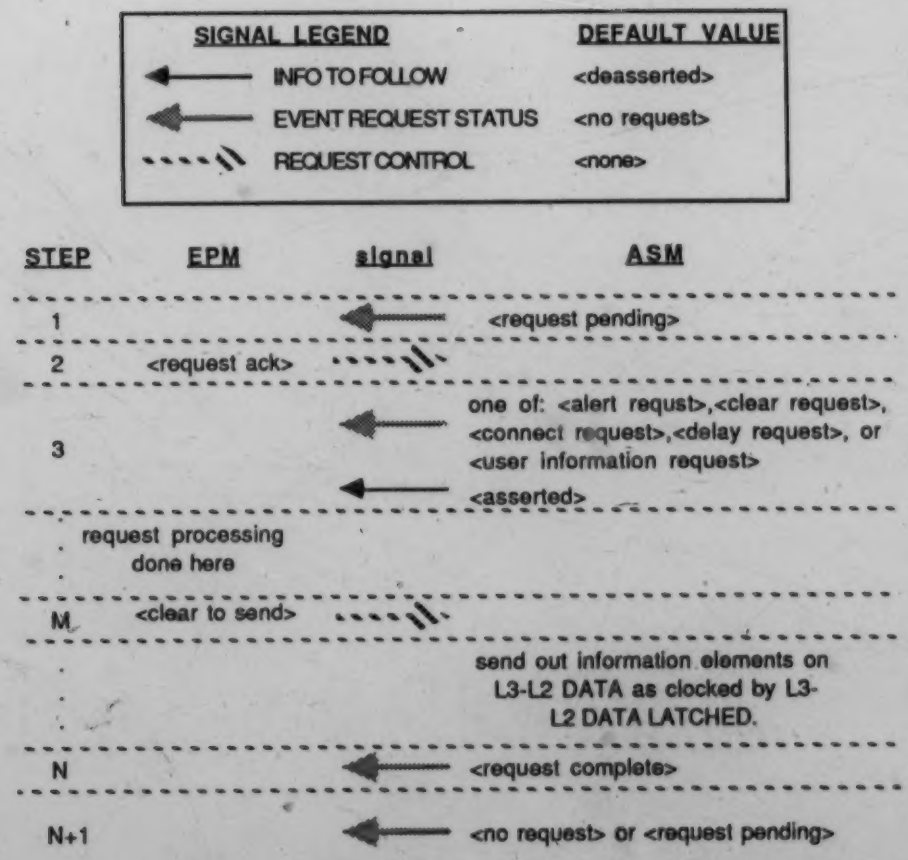


Fig. 4-9. ASM-EPM Handshaking - Information Elements to Follow

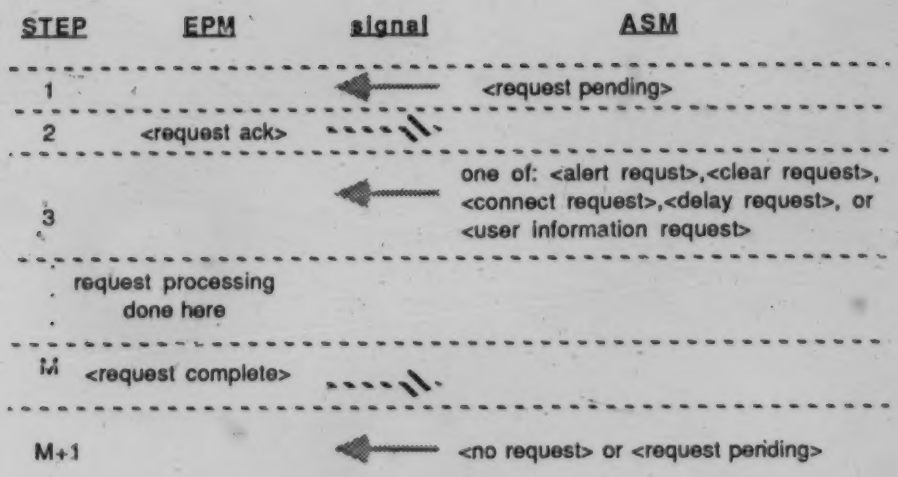


Fig. 4-10. ASM-EPM Handshaking - No Information Elements to Follow

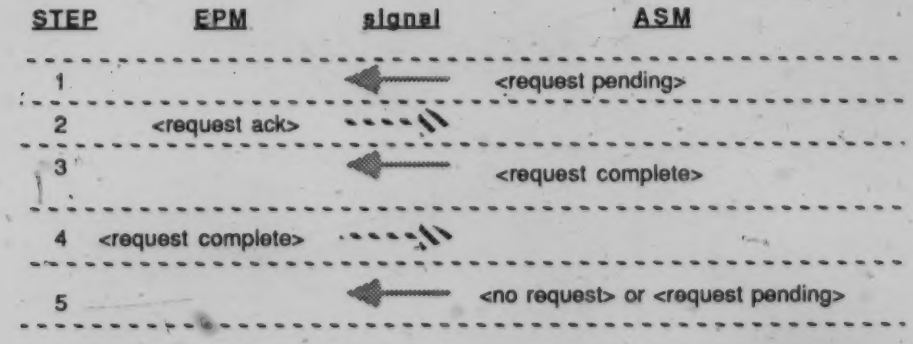


Fig. 4-11. ASM-EPM Handshaking - ASM Changes Its Mind

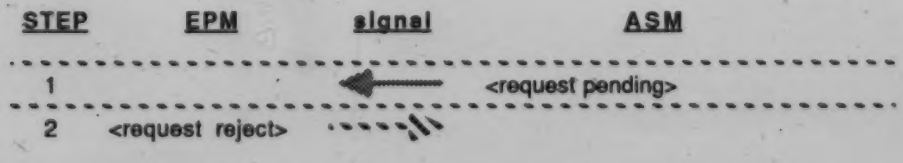


Fig. 4-12. ASM-EPM Handshaking - Illegal Request

4.6.4.2. The Event Queuing Technique

A reasonable design goal for the specification of the EPM is to create sufficient processing capability to handle heavy signalling activity without missing incoming messages or causing unacceptable delay in processing internal requests. Another design goal is to keep the EPM as simple as possible. Parallel processing of events relating to different call references is feasible because call references are mutually independent. However, a single system controller working on a serialized event stream is much less complex to implement. The emphasis then is to optimize the serial processing such that adequate throughput can be maintained at a minimum complexity.

Because there are actually three independent sources of events to be processed and all three can result in messages being sent out on the D channel it is necessary to queue up the events and handle them one at a time. Since the internal (system) clock of the protocol processor can be at a higher frequency than the "byte clock" of the D channel, if the processing of the incoming D channel is halted temporarily and the incoming bytes are buffered while another event is being processed, the EPM will be able to return to processing the D channel later and quickly "drain the buffer". Further, most messages that are received do not require that the EPM reply immediately. The reply, if any, is made after the ASM has initiated the appropriate internal request and the EPM has serviced it. Therefore, if a message is being sent out on the D channel the EPM can continue to service incoming messages. For that matter, if the outgoing D channel is buffered, the EPM could continue to process all events and simply queue up the outgoing messages in the transmit FIFO. The difficulties with this scheme are that buffer overflow must be avoided, and the messages initiated by the EPM must somehow be linked to the information elements that the ASM may want to add to them. Things are further complicated by the fact that the ASM may add an indeterminate number of bytes to the end of a message, thereby making buffering of information elements generated by the ASM very difficult to manage.

Before describing the event queuing and message buffering technique it is useful to provide a simplified diagram of the relevant signals used by the EPM, as shown in Figure 4-13.

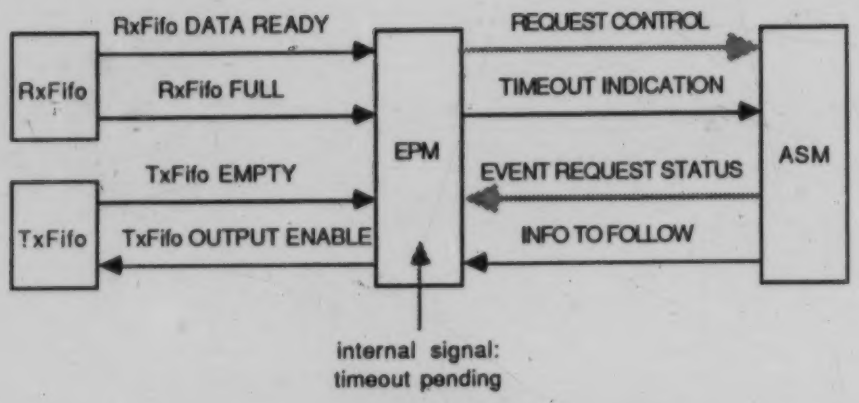


Fig. 4-13. Signals Needed for Event Queuing

The event queuing algorithm selected is based on the following restrictions:

1) The ASM-EPM interface restricts the ASM to asserting one internal request at a time. Further, all information elements to be added to the message by the ASM must be sent out on the L3-L2 DATA bus before the ASM may assert another internal request. In other words, the ASM does not buffer its outgoing information elements.

2) Only internal requests result in messages that may have information elements added to them by the ASM. Therefore, to simplify the linkage between a message initiated by the TxFifo and the information elements to be added to the end of it by the ASM, no internal requests are processed by the EPM unless the TxFifo is empty. In this way the EPM can push the message header onto the TxFifo, and then wait for the TxFifo to asserted TxFifo EMPTY before asserting <clear to send> to the ASM.

3) While the ASM is using the D channel, the TxFifo can buffer outgoing messages. Therefore incoming messages can continue to be processed. However, timeouts should not be processed until the ASM is given a chance to clear its internal request queue.

4) Internal requests have priority over incoming messages unless the RxFifo is getting full. This is not as strong as it sounds since the RxFifo FULL signal can be engineered to insure only a few messages are queued in RxFifo before RxFifo FULL is asserted.

Given these points, the event processing priorities can now be stated explicitly. In Table 4-5, the next event to process is defined as a function of the appropriate signals. Note that in the table, "T" represents true or asserted, "F" represents false or deasserted, and "X" represents don't care.

RxFifo READY	RxFifo FULL	TxFifo EMPTY	EVENT REQUEST STATUS	Timeout Pending	Event to process
F	F	X	not <request pending>	F	none
F	F	F	not <request pending>	T	none (TxFifo must be empty)
F	F	F	<request pending>	X	none (TxFifo must be empty)
T	F	F	<request pending>	X	incoming message
X	F	T	<request pending>	X	internal request
F	F	T	<no request >	T	timeout
F	F	T	not <no request> and not <request pending>	T	none (all internal requests must have been processed
T	F	T	not <request pending>	X	incoming message
T	T	X	<X>	X	incoming message

Table 4-5. Event Queuing Priority Table

If the EPM is processing an event, then that event must be "processed to completion" before the next event to process is selected, as defined below:

1) For a timeout event, "processed to completion" means the ASM is notified of the timeout and the appropriate message (if any) is pushed onto the TxFifo.

2) For an incoming message event, "processed to completion" is after the <eom> or <error> is popped from RxFifo, the ASM is passed the last byte of the message, and the reply message (if any) is pushed onto the TxFifo. In other words, once the EPM has begun to process a message it must wait until the final byte of that message has been processed before the processing of another event can begin.

3) For an internal request event, "processed to completion" means one of:

- The request is rejected by the EPM with REQUEST CONTROL = <request reject>.
- The ASM changes its mind by responding to the EPM's REQUEST CONTROL = <request ack> with an EVENT REQUEST STATUS = <request complete>.
- INFO TO FOLLOW was not asserted, the outgoing message has been pushed onto TxFifo, and REQUEST CONTROL = <request complete> has been asserted by the EPM.
- INFO TO FOLLOW was asserted, the first part of the outgoing message has been pushed onto the TxFifo, the TxFifo has sent out the message and replied with TxFifo EMPTY = <asserted>, the EPM has disabled the TxFifo's output with TxFifo OUTPUT ENABLE = <deasserted>, and REQUEST CONTROL = <clear to send> has been asserted by the EPM. Note that the EPM must insure that TxFifo is output enabled once the ASM has finished sending its information on the D channel and has asserted EVENT REQUEST STATUS = <request complete>.

We can now define precisely how the EPM sorts out which events to process. The following algorithm is the "heart" of the system controller function implemented by the EPM:

```
on power up or RESET do
    initialize all call states to null
    clear and stop all timers
    initialize all output signals appropriately
else when CLK2 goes high do
    restore (if only temporarily) any signals that may have been
        ...active last clock period:
        {
        RxFifo POP DATA          := <deasserted>
        TxFifo PUSH              := <deasserted>
        REQUEST CONTROL          := <none>
        MESSAGE INFO CONTROL     := <none>
        ELEMENT INFO CONTROL     := <none>
        TIMEOUT INDICATION       := <deasserted>
        }
    if ASM was using the D channel but now it is finished
        ...(i.e. EVENT REQEUST STATUS = <request complete>) then
        TxFifo OUTPUT ENABLE := <asserted>
    if previous active event has been processed to completion then
        select new active event (if any) according to table 4-5
    if there is an active event to process then
        if active event is incoming message then
            if RxFifo DATA READY then
                process the incoming message
        else (either request or timeout is active)
            process the active event
```

The final issue to analyze with respect to the buffering technique is the size of the transmit and receive FIFOs, where size means number of (data, type) pairs the FIFO can store. We can use RxMaxSize and TxMaxSize to mean the maximum size of RxFifo and TxFifo respectively. Further, we define RxFullSize as the number of (data, type) pairs RxFifo can buffer before RxFifo FULL is asserted. For convenience we also define RxOverflowSize = RxMaxSize - RxFullSize. The goal now is to determine what these values must be to insure neither the receive FIFO nor the transmit FIFO overflows during worst case conditions of signalling activity. Throughout these discussions it is important to remember that the D channel's bit rate will be equal in the two directions, so on average each time a (data,type) pair is pushed into RxFifo a (data,type) pair is popped from TxFifo (assuming of course that the data is available).

For the RxFifo the worst case scenario occurs when RxFifo is filled just below RxFullSize and the processing of an internal request with INFO TO FOLLOW = <asserted> begins. Since the EPM waits for the TxFifo to send out the header of the message before the internal request event is considered completed, RxOverflowSize must be greater than 5 since the maximum header size is 5 octets: protocol discriminator, call reference length, call reference (max. is 2 bytes), and message type. RxFullSize is selected to balance the processing between internal events and incoming messages under heavy load

conditions. An appropriate value cannot be determined here since the time the actual implementation takes to process events must be taken into account. In general, however, RxFullSize should not be greater than a few average length messages to ensure a burst of internal requests does not temporarily lock many messages in RxFifo. Regardless of the value selected for RxFullSize, as long as RxOverflowSize > 5 the RxFifo will never overflow.

For the TxFifo the worst case occurs when its output is disabled while the ASM sends out its information elements. This implies that TxFifo can be disabled for the time it takes to transmit the maximum message size (MaxMsgSize) minus the 5 message header octets that have already been sent out. MaxMsgSize might be as large as 256 or 512 bytes, although it can be restricted to a much smaller size if tighter limits are placed on how much user-to-user information is allowed in each message. While these MaxMsgSize bytes are being sent out on the D channel TxFifo will only have to buffer reply messages that are generated automatically by the EPM in response to incoming messages. The worst case occurs when the incoming message is MaxMsgSize long, every information element within it is the shortest possible (2 bytes), and every information element type is invalid. This causes the EPM to push a 13 byte status message onto TxFifo for every information element. Obviously this is not a very realistic scenario, but for what it is worth, if TxMaxSize > (MaxMsgSize + RxFullSize)/2 * 13 then the buffer will not overflow. Actually, TxFifo need not store the full contents of the Status message if it generates the message as needed and only stores the message type to send, call reference, cause value, and diagnostic value. In this case the storage requirements are approximately double MaxMsgSize+RxFullSize. In either case there is a known size that can be selected to guarantee TxFifo never overflows, even under worst case error conditions.

4.6.4.3. Functional Description of the EPM

Up to now we have covered the EPM's interfaces to the two FIFOs and the ASM, the EPM's event queuing and message buffering technique, and the system control loop executed by the EPM. Indeed all that remains is for us to specify exactly what must be done to process the three types of events: timeout, internal request, and incoming message. The algorithms for processing each type of event are given below. It should be remembered that these algorithms fit within the "process active event" clause of the system control loop. For internal request and incoming message events the processing may require several CLK2 cycles before it is completed and the control loop is permitted to select a new active event. This is noted by an internal variable called

StillProcessingEvent which we should assume has been asserted by the active event selection process and will remain asserted until the appropriate event processing algorithm deasserts it. The system control loop executes the appropriate event processing procedure every CLK2 clock cycle until it is completed.

Timeout processing is quite simple. The timeout pending internal signal is asserted when one or more of the call reference's timers have expired. One of these call references is selected for timeout processing. Let us define TimeoutCallRef as the value of the call reference with an expired timer, and TimeoutCallState as the call state of that call. The processing algorithm is as follows:

```
Perform appropriate actions for the TimeoutCallState according to
    ...the clear request processing defined in the SDL diagram:
    {
    if there is a Disconnect or Release message to send then
        TxFifo PUSH := <asserted>
        TxFifo INFO TO FOLLOW := <deasserted>
        TxFifo MSG TYPE := <release> or <disconnect> as appropriate
        TxFifo CALL REF := TimeoutCallRef
        TxFifo MSG CONTENTS := cause value of 111
    Reset or Stop the call's timer as appropriate
    Update TimeoutCallRef's call's state
    }
StillProcessingEvent := <deasserted>
notify the ASM of the timeout:
    {
    TIMOUT INDICATION := <asserted>
    EPM-ASM CALL REFERENCE := TimeoutCallRef
    CALL STATE := new call state of TimeoutCallRef
    }
```

The implementation of internal request handling is complicated somewhat by the handshaking performed between the EPM and the ASM, as was explained previously. In order to properly specify how this handshaking can be implemented it is necessary to define a processing state variable we shall call RPState (for Request Processing State). RPState can take on the value <new request>, <getting request>, or <hand off to ASM>. The active event selection process initializes RPState to <new request>. The processing algorithm is as follows:

```
if RPState is <new request> then
    REQUEST CONTROL := <request ack>
    RPState := <getting request>
    StillProcessingEvent := <asserted>
else if RPState is <getting request> then
    }
    (the ASM is required to set EVENT REQUEST STATUS immediately)
    if EVENT REQUEST STATUS is <request complete> then
```

```
        REQUEST CONTROL := <request complete>
        StillProcessingEvent := <deasserted>
    else (alerting, clear, connect, delay, or user info request)
        {
        if request is NOT valid (as defined by SDL diagram) then
            REQUEST CONTROL := <request reject>
            StillProcessingEvent := <deasserted>
        else
            get the call reference from ASM-EPM CALL REFERENCE
            if the request is <connect request> and
                ...the call reference is 0 (i.e. setup request) then
                allocate a new call reference
                inform the ASM of the new call reference by setting
                    ...EPM-ASM CALL REFERENCE appropriately when REQUEST
                    ...CONTROL is set to <clear to send> later on
            as appropriate do the following:
                {
                push the proper outgoing message into TxFifo
                change call's state
                stop or reset the call's timer
                }
            if the ASM will be adding information elements to the end
                ...of the message then
                RPState := <hand off to ASM>
                StillProcessingEvent := <asserted>
            else
                REQUEST CONTROL := <request complete>
                StillProcessingEvent := <deasserted>
        }
    }
else (RPState is <hand off to ASM>, so EPM waits for TxFifo to clear)
    if TxFifo EMPTY is <asserted> then
        TxFifo OUTPUT ENABLE := <deasserted>
        REQUEST CONTROL := <clear to send>
        StillProcessingEvent := <deasserted>
    else
        StillProcessingEvent := <asserted>
```

The incoming message event is the still complex to implement, even given the simplifications made to the I.451 protocol. There are two key function for which processing state variables are required. These are:

1) MessageState - Keeps track of where the EPM is with respect to the overall processing of the message. It can take on the values <protocol discriminator>, <call reference length>, <call reference byte 1>, <call reference byte 2>, <message type>, <processing information elements>, and <skipping message>. When the event selection function determines that an incoming message is the next event to process, MessageState will be initialized to <protocol discriminator>. The call reference could be 1 or 2 bytes. If the protocol has been selected such that only a 1 byte call reference is valid, a message with a two byte call reference is ignored. If

a 2 byte call reference is valid, then a 1 byte call reference should still be accepted. Because our simulation was set up for a 1 byte call reference that is what is shown in the algorithm below.

2) ElementState - Keeps track of where the EPM is with respect to the overall processing of the information element. It can take on the values <element type>, <element length then process>, <element length then skip>, <processing>, and <skipping>. ElementState is initialized to <element type> after the message type has been processed, and after every information has been processed.

Given the definition of these internal state variables the incoming message processing appears as follows:

```
case MessageState of
    }
    <protocol discriminator>:
        if RxFifo DATA TYPE is <som> and RxFifo DATA is the
            ...protocol discriminator value (00001000) then
            MessageState := <call reference length>
        else
            MessageState := <skipping message>
    <call reference length>:
        if RxFifo DATA TYPE is <msg> and
            ...RxFifo DATA is 1 then
            MessageState := <call reference byte 1>
        else
            MessageState := <skipping message>
    <call reference byte 1>:
        if RxFifo DATA TYPE is <msg> then
            MessageState := <message type>
            at this point the message's call reference is
                ..."internalized", or made relative to this side's
                ...view of things. In other words,
                ...the high order bit or direction flag bit
                ...of the call reference is toggled because the two
                ...ends of the link view the direction of the call
                ...in the opposite way.
            remember the call reference, get its call state from the internal registers
                ...used to store call information
    <message type>:
        if RxFifo DATA TYPE is <msg> or <eom> then
            (note that RxFifo DATA is the message type)
            if the message type is unknown type then
                signal TxFifo to send out a Status message: cause
                    ...value is 97, diagnostic value is the
                    ...message type, call state value is the call
                    ...reference's local call state
                MessageState := <skipping message>
```

```
        else if the call reference is 0 and it is not a Status message then
            signal TxFifo to send out a Status message: cause
                ...value is 81, diagnostic value is the
                ...call reference, call state is null
            MessageState := <skipping message>
        else if it is a Setup message then
            if direction flag is 0 (remember bit is flipped already) then
                MessageState := <skipping message>
            else if call state is null then
                MessageState := <processing information elements>
                pass the ASM the  message type and call reference
                MESSAGE INFO CONTROL := <som>
            else
                MessageState := <skipping message>
        else if the message type is illegal given the call state
            ...of the call reference then
            signal TxFifo to send out a Status message: cause
                ...value is 101, diagnostic value is the
                ...message type, call state value is the call
                ...reference's local call state
            MessageState := <skipping message>
        else if the call state is null and the message type is
            ...not Setup or Status then
            if the message type is not Release Complete then
                signal TxFifo to send a Release Complete with
                    ...a cause value of 81
            MessageState := <skipping message>
        else
            pass the ASM the  message type and call reference
            if RxFifo DATA TYPE is <eom> then
                perform the same end of message processing that is
                    ...done in the <processing information elements>
                    ...procedures below
            else
                MessageState := <processing information elements>
                MESSAGE INFO CONTROL := <som>
<processing information elements>:
    {
    if RxFifo DATA TYPE is <error> then (notify ASM of bad msg)
        MESSAGE INFO CONTROL := <msg bad>
    else
        {
        using the ElementState variable, cycle through the
            ...information elements contained in the message and
            ...pass them to the ASM using the appropriate signals.
            ...If an element is not of a valid type, then send out
            ...the appropriate status message and just skip over
            ...it without passing it to the ASM. If the message
            ...type is Release or Disconnect then save the cause
            ...value from Cause information element for the
            ...Release Complete reply message or the Release
            ...reply if the call state is 11.
```

```
                ...If the message is Status then save the call state
                ...value from Call State information element just in
                ...case this side's call state for the call reference
                ...is null and a Release Complete message needs to be
                ...sent out because the Status message's call state
                ...value was not null.
            if RxFifo DATA TYPE is <EOM> then
                If there are missing mandatory elements or
                    ...the last info element is incomplete then
                    send out appropriate Status message
                    MESSAGE INFO CONTROL := <msg bad>
                else
                    as appropriate do
                        update the call state and timer
                        send a reply message
                    MESSAGE INFO CONTROL := <msg good>
            }
        }
    <skipping message>:
        (do nothing)
    } (end of case statement)
if RxFifo DATA TYPE is <eom> or <error> then
    StillProcessingEvent := <deasserted>
else
    StillProcessingEvent := <asserted>
RxFifo POP DATA := <asserted>
```

4.6.4.4. Data Structures for Call Management

In the previous sections there has been no mention of the techniques available to implement the the call state registers and timers required for the call references. The single largest problem is the very large number of potential call references that must be supported when a two byte call reference is used: 65536 to be exact. Even with a one byte call reference significant complexity is added when an individual timer and call state register is provided for each of the 256 call reference values. Therefore, unless the I.451 recommendation is amended to permit implementations to greatly restrict the number of valid call references, the EPM hardware will have to perform some form of hashing function or look up procedure in order to associate each call reference with a timer / call state register pair from the shared pool. In this way far fewer timers and registers are needed. This type of approach can be implemented in any number of ways, and is similar to many classic resource sharing problems such as dynamic memory allocation or buffer pool allocation. For simplicity, the simulated implementation uses a one byte call reference and all 256 potential values are given an individual register and timer. This may actually be the least complex solution for some implementations, but a two byte call reference should not be implemented in this way. It would be a great waste

of hardware to provide for 65536 active call references when only a few will be active on the link at a time.

With the subset I.451 protocol a 3 bit (8 state) call state register and a 2 or 3 bit timer is required for each call reference that is not in the null state. The individual timers need not have a very high resolution since the timeout is only necessary to recover from error conditions. If the timeout period is say 30 seconds then a timer resolution of 5 seconds is probably all that is needed. Having a coarse timer granularity will also minimize the processing load required to maintain the timers. Other storage elements may also be required on a "per call reference" basis to support the resource sharing method used in the implementation.

4.6.5. The ASM

Other than stating that the ASM must support the handshaking procedures specified for the EPM-ASM interface, there is not much more to be said about the requirements for the ASM until a specific application is selected for implementation. The nature of the ASM depends on the ultimate source of the signalling information. If the application is a user terminal, then the human is the final source (and sink) of the signalling information. If the application is a switch, then the ASM may be acting mainly as a temporary storage and routing device for the signalling information. In either case, the ASM must be capable of accepting incoming messages from the EPM at the full D channel rate. How these messages are stored and processed is up to the ASM. If the "stream processing" philosophy of the rest of the protocol processor can be maintained in the ASM implementation then the overall throughput of the signalling system will not be hampered by processing delay. In fact it would be the bandwidth of the D channel and the response time of the ultimate source of the signalling information that would determine the signalling system's capabilities. Without significant processing delay slowing things down, there is good potential for creating an ISDN signalling system that supports extremely quick service establishment and disestablishment. While this will probably not shake up the voice services market much, in the data services market it could make new ISDN based circuit switched data services strong competitors to the older packet based services.

Chapter 5

Software Simulation of the Protocol Processor

5.1. Chapter Overview

In this chapter the focus shifts from the functionality and implementation aspects of the protocol processor to the software simulation of the protocol processor hardware. Of course in reality the research was not cleanly split between developing the specification for the protocol processor's implementation and developing the simulation of that implementation. There were refinements made to the original specification, especially in the definition of the interfaces between modules and in the timing technique used. In this respect, the simulation was extremely valuable in highlighting the important details that must be taken into consideration when the actual hardware implementation is attempted.

The simulation is a functional simulation of the protocol processor specified in the previous chapter. Although the interfaces and timing between modules and the functionality of each module is accurately simulated, the simulation makes no attempt to emulate a circuit level implementation of the protocol processor. The simulation hardware configuration consists of two IBM compatible Personal Computers (PCs) communicating with each other through a standard RS-232 serial interface. The simulation software is written in Microsoft Pascal with a small amount of Microsoft assembler to interface to the low level hardware. Each PC runs a protocol processor simulation program with the serial link being used as a D channel. The human interface of the simulation allows the user to control and exercise what would normally be the PSM-ASM interface, so in effect the user takes the place of the ASM. Special user interfaces were provided to monitor the internal status of the various "hardware" modules and to create error conditions to test the error handling procedures of the subset protocol and the protocol processor.

The chapter opens by discussing why the simulation was a valuable stage in the research. This is followed by a description of the key features and structures of the simulation software, with special emphasis on the technique used to simulate the hardware timing. Next the simulation's user interface and its logical relationship to the protocol processor is discussed, followed by a description of the implementation and testing procedures used throughout the software development cycle. In the final section, the results of the simulation are analyzed with respect to the feasibility of implementing the protocol processor in hardware.

The full source listings for the simulation software can be found in [Brown][16].

5.2. Benefits of a Functional Simulation

The simulation provides important information concerning the following areas:

1) It confirms that the simplifications proposed for the network layer protocol do create a valid subset of the I.451 protocol.
2) It resulted in refinements being made to the inter-module interfaces and timing. Clean definitions of interfaces is desirable regardless of implementation technology.
3) The software source code creates an extremely detailed functional specification for the protocol processor. Many nuances of the functional specification became evident during the coding and testing of the simulation software.
4) It provides a very good measure of the "implementation complexity" of the major modules of the protocol processor.

Although it is true that a gate level simulation of a hardware implementation of the protocol processor could have provided the same information and more, the resource and time requirements of such an approach were not compatible with a MSc program. It should also be mentioned that the support software used in the simulation creates a good base for future simulations of ISDN or similar protocol implementations. Further, if a hardware prototype of the protocol processor is developed it will require a test system to exercise it. If the prototype is designed to interface to the PC's backplane then the support software can easily be modified to act as the test system's user interface.

5.3. Structure of the Simulation Software

The simulation software is divided into three major functional blocks: the hardware simulation module, the user interface modules (3), and the simulation control module. The hardware simulation module is structured to mimic the hardware partitioning proposed in the previous chapter. The user interface modules provide the human with the input and visual displays necessary to monitor and control the execution of the hardware simulation code. The simulation control function ties everything together by providing timing and calling sequence control for the hardware simulation and user interface functions. Graphically the overall structure of the simulation program appears in Figure 5-1. The various modules, data flows and control flows are defined in Tables 5-1 to 5-3.

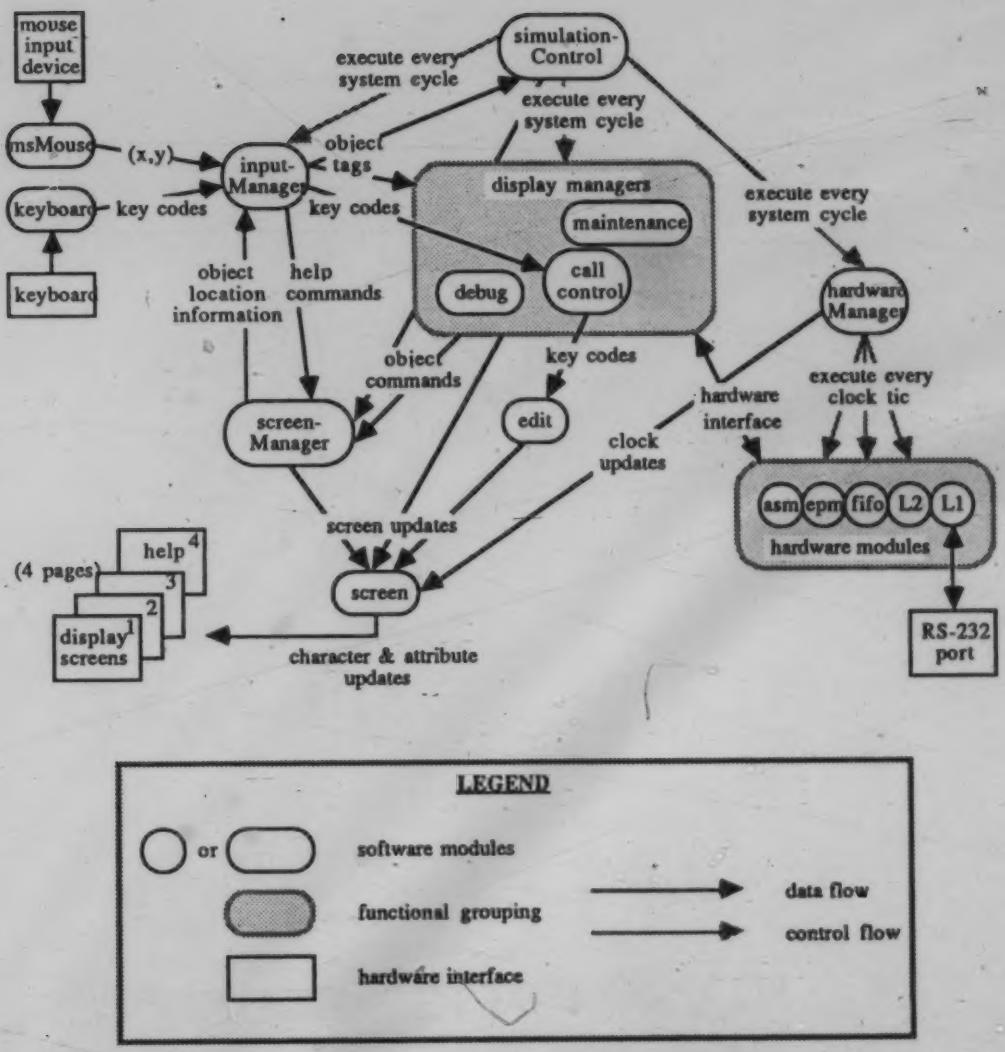


Fig. 5-1. Structure of the Simulation Software

module name	description

simulationControl	This module mainly consists of an control loop which controls the execution of the rest of the simulation. The normal control cycle is as follows:
	repeat
	call input manager to check for mouse input
	if there was input then
	check if input changes global settings such as visible
	...display page, hardware clock enable, etc.
	call the active (visible) display manager to process
	...input and perform appropriate actions
	if the hardware clock is enabled then
	call hardware manager
	call each display manager to update its screen

mouse input device	When the user clicks the mouse button the coordinates of the mouse pointer's position on the CRT screen are stored in a special register and a flag is raised.
msMouse	This is the Microsoft mouse driver plus a small amount of control software used to "debounce" the mouse clicks.
keyboard (device)	The standard MS-DOS BIOS keyboard interface is used.
keyboard (module)	This module interprets the key codes in order to pre-parse the edit keys.
inputManager	The mouse and keyboard routines are called by the input manager on every system cycle. If a mouse click has occurred the (x,y) coordinates are used to look up the object associated with those coordinates. Obviously, only the visible display page's object table is used during the look up procedure. The objects identification tag is passed to the simulation control and display managers for processing.
display screens	The PC is used in CGA text mode, so 4 independent full screen pages are available, with each page containing 80 columns by 25 lines of addressable characters. Only one page can be active at a time, but the information on the invisible pages is not lost. Switching between pages requires a single register to be changed. The PC hardware permits direct software access to the separate video RAM from which the CRT screen is refreshed. The visual content of each character position is determined by setting two bytes in the video RAM: the character's 8 bit code (extended ASCII) and 8 bit color attribute.
screen	This module provides basic screen and object utilities to the rest of the software. Because the standard IBM BIOS screen access routines are so slow they are not used; the CGA card is written to directly.
edit	This module provides a fairly complete "window editor" where the window is some portion of the visible screen.
screenManager	This module provides some higher level screen and object utilities to simplify common functions such as display page initialization, object definitions, etc. The help function is also implemented by this module.
display managers	Each display page has a software module to look after it. The module is responsible for keeping the information on the display up-to-date, and for performing the actions associated with the objects selected by the user.
maintenance	see User Interface subsection.
debug	see User Interface subsection.
call control	see User Interface subsection.
hardwareManager	The main purpose of this module is to keep track of an internal real time clock used to time the simulated hardware clock. The hardware clock's frequency is determined by a user adjustable constant defined in seconds and 1/100 seconds. On every clock tic (i.e. every rising edge of the simulated hardware clock) the hardware simulation code is executed. The details of the hardware timing simulation are discussed below.
hardware modules	This code is partitioned into L1, L2, fifo(RxFifo&TxFifo), EPM, and ASM modules. The interfaces between modules and the functionality of each module is defined in the previous chapter.

Table 5-1. Simulation Software Module Definitions

data flow name	description
(x,y)	Screen coordinates of the latest mouse click position.
key codes	The standard IBM keyboard character codes are used. Additionally, keys are classified according to their use.
object tags	The input manager uses the object look up table associated with the visible page to associate a mouse click with an object tag. The tag (the number) of an object that has been "clicked on" is passed to the visible page's display manager for processing.
object location info	The (x,y) coordinates are used to locate the object (if any) at the selected location.
help commands	The inputManager detects when the help object is selected and calls the screenManager directly to display the help information associated with the visible page.
object commands	There are a few "universal" objects available on every display page used for changing global settings such as selecting the visible page, enabling the hardware simulation clock, and ending the program. Selection of any of these objects is processed by the simulationControl module. All other objects are controlled by their appropriate display manager.
screen updates	Calls into the screen utilities module to update a display page's contents, or modify an object.
clock updates	There is a dual clock display maintained in the upper right corner of the visible display page. The left clock is updated every hardware clock cycle, the right clock every system cycle.
character&attribute updates	Screen updates are made on a character by character basis by writing a character code or attribute (or both) into the video RAM. The visible page is set with a single write to the CGA card's display page register.
hardware interface	This data flow consists of shared variables rather than the actual movement of data. When a display manager wishes to read or modify any of the signals or internal register used by the hardware simulation code it simply reads or writes the global variable used to represent that signal or register.

Table 5-2. Simulation Software Data Flow Definitions

control flow name	description
execute every system cycle	The simulationControl module's control loop runs continuously, and without using any real time clocking function to throttle its cycle frequency. Therefore, the control loop's cycle frequency depends on the time it takes the called routines to execute.
execute every clock tic	The hardwareManager uses a real time clock to generate a simulated hardware "clock tic" at a user selected frequency. The hardware simulation code's execution is throttled by this clock since the hardware manager only calls the hardware simulation code once per clock cycle.

Table 5-3. Simulation Software Control Flow Definitions

5.4. Hardware Simulation Control and Timing

As was noted in the previous section, the hardware manager is responsible for timing the execution of the various software routines that simulate the functionality of the protocol processor's hardware modules. The hardware manager is called on every system control loop cycle. It executes the following algorithm:

```
if the current time is greater than DeltaClock then
    {
    (execute CLK1 clocked modules)
    Process L1
    Process RxFifo
    Process TxFifo
    Process ASM
    Output L1
    Output RxFifo
    Output TxFifo
    Output ASM

    (execute CLK2 clocked modules)
    Process L2
    Process EPM
    Output L2
    Output EPM

    DeltaClock := current time plus hardware clock width
    }
```

There are several points to note concerning the above algorithm:

1) The execution of each hardware simulation module is separated into two portions, the Process subroutine and the Output subroutine. This mimics a clocking scheme in which every module latches its inputs on the rising clock edge, and changes its outputs on the trailing clock edge. Strictly speaking, the two-phase clocking scheme makes synchronizing the output signals unnecessary. However, the scheme was used to aid in debugging the simulation software since it creates a clear separation between the processing phase and the output modification phase.

2) The two-phase clocking scheme is implemented by grouping and staggering the subroutine calls as shown.

3) The interface signals between modules were implemented using global variables. Given the serial nature of the simulation's implementation there was no requirement for synchronizing variable access other than that already provided by the two-phase clocking scheme.

4) The Process ASM and Output ASM modules have not been previously defined. During the initial testing the ASM simulation module was effectively null since the user interface was used to provide all ASM functionality. During the latter stages of the simulation a more comprehensive ASM module was written to automatically perform the handshaking required by the ASM-EPM interface. This ASM also supplies "dummy" mandatory information elements for those messages that require them. The user interface (via the call control display manager) is used as the source of all internal requests.

We have been restricting our discussion to the internal timing of the simulation as it executes on a single PC. It is equally important to synchronize the timing between the two PCs as they exchange D channel information. As was noted in the previous chapter, ISDN transmission links normally have the two directions of the D channel running at the same bit rate and in a synchronous fashion. In order to simulate these D channel properties the L1 modules exchange D channel data over the RS-232 link using a fully locked rendezvous technique. The inter-PC cabling and pinouts are shown in Figure 5-2.

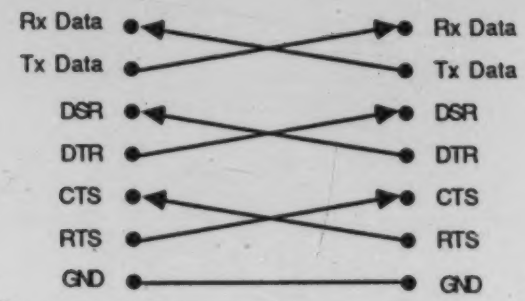


Fig. 5-2. Inter-PC Wiring and Pinout

The D channel's byte rate was assumed to be at a lower frequency than the protocol processor's system clock. A clock rate ratio of 1:4 (D channel byte rate to system clock) was selected for the simulation. In an actual implementation the clock ratio will be a function of implementation technology and the D channel's bit rate. The L1 module satisfies the clocking ratio and inter-PC synchronization requirements using the algorithm shown below. The internal counter used to count hardware clock cycles is initialized to a value of 4. The following algorithm executes each time the hardware manager calls the Process L1 subroutine:

```
L1-L2 DATA READY := <deasserted>
L2-L1 DATA BUS LOCK := <deasserted>
L2-L1 DATA LATCHED := <deasserted>
cycle counter := cycle counter - 1
if cycle counter is 1 then
    L1-L2 DATA BUS LOCK := <asserted>
else if cycle counter is 0 then
    {
    L1-L2 DATA READY := <asserted>
    L2-L1 DATA LATCHED := <asserted>
    cycle counter := 4
    clear the Rx register of any garbage data
```

```
assert the DTR control line
wait for DSR to become asserted
transmit L2-L1 DATA TYPE using the Tx side of the serial port
wait until the Rx side of the serial port is receive ready
read the Rx side of the port, place the byte in TempStore, save the return status
assert the RTS control line
wait for CTS to become asserted
transmit L2-L1 DATA
wait until the Rx side is receive ready
read the Rx side of the port and place the byte in L1-L2 DATA, save the return status
if either Rx status shows an error has occurred then
    L1-L2 DATA TYPE := <error>
else
    case TempStore of
        0 : L1-L2 DATA TYPE := <none>
        1 : L1-L2 DATA TYPE := <som>
        2 : L1-L2 DATA TYPE := <msg>
        3 : L1-L2 DATA TYPE := <eom>
        otherwise: L1-L2 DATA TYPE := <error>
}
```

In the simulation code the updates to the output signals are not made until Output L1 is called. Notice that the double control line handshake ensures the two PCs rendezvous and exchange two bytes every four hardware clock cycles. This is true regardless of the system cycle time or even the hardware clock's cycle time. In an actual implementation the system clock can be decoupled from the D channel byte clock as long as the timing synchronization boundary problem discussed in the previous chapter is handled appropriately.

5.5. The User Interface

The simulation's user interface is almost entirely mouse driven. On every CRT display page there are short sections of text which are used as selection objects. The text is normally a few words that inform the user of the object's function should it be selected. Selected objects are highlighted to appear different from non-selected objects, but the selected object's text does not change. The final (and most complete) simulation uses a "call control" display page, a "maintenance" page, a "debug" page and a "help" page. The first three displays are controlled by individual display managers, where each manager is responsible for keeping its display up-to-date with whichever global variables it uses to report on the status of the hardware simulation modules. The display managers also modify the ASM interface and other internal variables as requested by the user.

The call control display page provides the user with a fairly high level view of the status of the first ten call references (in both directions). Using this page the user interacts with the ASM module using internal requests such as alerting, connect, etc. The

requests are then translated by the ASM to control the ASM-EPM interface. This display is the only one which requires keyboard entry, and even then it is simply for entering the user-user information element. All other inputs to the system are mouse driven. The maintenance display page allows the user to modify certain internal registers used for timing. As well, direct modification of the status of the first ten call references is available using this display. The debug display page provides the user with extremely detailed information on many aspects of the hardware simulation. As well, the bit level access to the D channel provided by this display permits the user to force errored messages to be transmitted. This was extremely useful in testing the error handling capability of our implementation of the I.451 protocol. All three pages access to the two non-overlapping internal hardware clocks so that they may be stopped, pulsed once (each of the two clocks is pulsed once), or left free running at the rate defined by the maintenance display.

The help display page provides the user with quick access to help information. This display page is unique in that it it does not require a display manager to control it since it is built into the general screen support software. Each display manager provides the help facilities with a file containing the help information to display should the user select the "HELP" object field in the lower left corner. The help information is shown on a separate screen page so the original display is not disturbed. When the user has read the help information a "RETURN" object field is selected. Since the help information file name can by changed at any time by the active display manager, the available help information can be changed dynamically depending on the state of the system. The help facilities were not used extensively by the simulation system, but they are general purpose enough to enhance the overall power of the user interface.

5.6. Coding and Testing Procedures

The design technique used to develop the simulation software is probably best described as "rapid prototyping" using "incremental development."[17] This technique was used for several reasons:

1) Hardware details of the PC were not well understood before the project began. A somewhat "bottom-up" approach was needed initially.
2) The goal was to improve the original protocol processor specifications during the implementation of the hardware simulation. An iterative and incremental system development encouraged this process.
3) In a multi-person project, functional specification and interface instability is a recipe for disaster. However, in this project the programming was done by one

person so there was little difficulty in making frequent changes to test out new ideas.

From a software engineering standpoint, rapid prototyping and incremental development flies in the face of much of the classical "top-down" structured design techniques. In terms of software quality and overall time of development, incremental development seemed to be as good or better than one would expect of the more rigorous structured design techniques. With respect to the implemented functionality meeting actual requirements (as opposed to perceived requirements), rapid prototyping wins hands down. But of course the nature of the project was almost perfectly suited to the approach taken. With a multi-person development team the success of this approach would depend almost entirely on establishing and maintaining excellent communications among the team members. However, in a situation with poorly defined or changing system requirements this type of approach should be considered.

The software testing procedures used reflect the incremental nature of the software development. As the various modules were written they were tested individually, and then integrated with the progressively larger set of existing modules. Once the general software was debugged the testing turned to the hardware modules. The physical and data link layers presented no significant testing problems, so the focused quickly shifted to the network layer protocol and its implementation. The call control display page provided a good "high level view" of the internal states of the call references, while the debug display page provided a detailed view of the internal operations of the PSM, and permitted bit level access to the D channel. The combination of these two displays, plus the ability to tightly control the simulated hardware clocks, created an environment in which studying the subset protocol "in action" proved to be quite straightforward. Some minor changes in the initial implementation of the PSM were made to overcome interfacing problems between the FIFOs and the EPM module. Other changes arose when the August '87 version of I.451 was released. New error condition handling procedures were adopted and the symmetric implementation was slightly different than we had used originally.

Given the amount of work put into the specification of I.451 by the CCITT, it is no surprise that the overall structure and error handling procedures of the network layer are very well behaved. Because the majority of the general and error handling procedures are defined (and implemented) independently of the call reference's call state, the testing of these procedures is quite manageable - there is no need to test every error condition in every possible state. Even testing the validity of the SDL diagram is not overly complex since, by definition (and implementation), the procedures executed

within each state are independent of the path into that state. In summary, although the testing of the network layer's call control procedures for a single call reference was time consuming, there where no significant problems in completing a very thorough test of both the protocol and its implementation.

When multiple calls are active, the call reference structure insures that the protocol cannot create interaction between calls. Therefore, the testing of the multiple call scenario was reduced to insuring that the buffering and event selection process used by the PSM resulted in the correct actions being taken. This was the most complex aspect of the testing procedures since it was necessary to simultaneously track the three sources of events: incoming messages, internal timeouts, and requests from the ASM. With its ability to display all the critical internal registers of the PSM, the debug display screen proved invaluable during this portion of the testing. In general, the testing procedure was to create the internal conditions necessary to test the implementation of the event selection process as defined by the *Event Queuing Priority Table* (Table 4-5). To simplify the process, a short RxFifo was used to insure the RxFifo EMPTY signal could be toggled without requiring a large number of messages to be created.

The coding and testing of the simulation system revealed some interesting aspects of simulating a hardware oriented signalling system protocol processor using a general purpose high level language and multiple computers. The majority of the effort was spent setting up and testing the "utility modules" such as the user interface and the inter-computer link. This was complicated somewhat by the desire to create a general purpose protocol simulation (and testbed) system, so the assumption is that using the system to simulate a different implementation or protocol would proceed much more quickly. Even so, the advantages of having a good user interface certainly outweighs the effort required to create it. As a general observation, it is not difficult to structure the simulation software to mimic the hardware interfaces and partitioning. The inter-module timing can be simulated using a software module calling strategy based on a hardware clock analogy. Restricting the use of global variables to insure each module only has access to its specified "I/O lines" maintains the integrity of the simulation, while liberal use of global read only variables simplifies debugging and testing. Tight control of the hardware clocks gives the user excellent execution control.

5.7. Hardware Implementation: Is It Feasible?

It was concluded in the previous section that the simulation successfully tested the functionality of the subset I.451 protocol and the proposed implementation of the

protocol processor. What is left to discuss is the larger issue of whether or not hardware implementation of the ISDN network layer is feasible. Unfortunately, the first step must be to restrict the analysis to the PSM since no ASM was implemented. To answer the feasibility question, we need to study the nature of the PSM's code and decide what the best hardware implementation would be, and whether the complexity of this implementation is reasonable in today's technology.

The next step in the analysis is to remove the RxFifo and TxFifo from consideration. The reason for this is clear, the issue of whether layer 2 passes information up to layer 3 via a stream of bytes or a pointer to a complete packet should only be an implementation detail when compared to the larger issue of whether the functionality of the PSM (as contained in the EPM) can be implemented in hardware. Although it is "taken out of context", the source listing of the EPM simulation module is shown in Appendix D. From the code it can be seen that a significant nesting of state variables is required to maintain processing continuity as the incoming message is processed. If the message were to arrive as a complete packet, these state variables could be replaced by nested control structures, but the complexity of the implementation would remain about the same. This also applies to the interface between the EPM and the ASM: state variables are required to track the handshaking between these modules, but if the stream oriented interface was replaced with a packet oriented interface the complexity would not change significantly. Therefore, the EPM code provides a reasonable measure of implementation complexity, and further, the nature of the code provides a clue as to the type of processing required.

The areas of complexity that dominate the EPM functionality are:

- Call reference state table management.
- Timer facilities.
- I/O to other hardware modules.
- Flow-control as a function of call state, message contents, and I/O lines.

Of these, the call reference state table will create the largest RAM requirements, especially if a two byte call reference is supported. Timer facilities could introduce significant complexity if there is a need for fine grained timing on a per call basis, but we have shown that a coarser grained timer is sufficient. The I/O is not overly complex, with the PSM-ASM handshaking requiring the most sophisticated interface. The flow control will probably be the source of the greatest complexity, but there is nothing in the EPM code that would lead one to believe that a "custom computer on a chip" approach would not suffice. Even if a more complex protocol were to be implemented, the basic structure of the control flow need not change. If the control flow were table driven, the

ability to adapt to different versions of I.451 could be incorporated in the design. There may be a need to go to a multi-chip solution, with the three major modules being the data-store RAM, a firmware ROM, and a protocol processor built as a custom computer. It should also be noted that the EPM's code uses very few instructions types, with comparisons, flow control, and assignments (to internal variables and I/O lines) dominating. This type of application may be well suited to a Reduced Instruction Set Computer (RISC) approach in which the microprocessor is greatly simplified by removing instructions that are not heavily used by the application.

In summary, it is probably safe to say that there is a technology mismatch between the network layer protocol and a hardware implementation if the hardware is restricted to combinational logic and simple controllers. However, with modern VLSI technology and good computer aided engineering tools it would certainly be short sighted to assume that there is no place for hardware orientation in the network layer implementation.

Chapter 6

Conclusions and Future Work

6.1. Chapter Overview

In this chapter we re-visit the key results of the research in order place them in the overall context of ISDN standards and hardware implementation technology. There are essentially three areas in which the research produced results:

1) *Standards*: modelling techniques, allocation of functionality among the OSI layers, and symmetric signalling protocols.

2) *Implementation*: stream processing, and layer 3 partitioning.

3) *Simulation techniques*: functional modelling using a general purpose high level language, and user interfaces.

In the next three sections, the key results in each of the above areas are summarized. The final section of this chapter describes the areas where further work is needed.

6.2. Results Related to ISDN Standards

The models developed for symmetric ISDN transmission links and for the signalling system's protocol processor aid greatly in understanding the ISDN protocols and their processing requirements. Using a symmetric link model, and partitioning the network layer into a protocol specific module (the PSM) and an application specific module (the ASM) makes the ISDN signalling protocol understandable without resorting to the more complex model of a circuit switched telecommunications network with a differentiation between the "network side" and the "user side". Further, the link model can be extended to model a circuit switching network, and this extension is made without reference to a network side or a user side. The value of these models would be independent of the detailed nature of the ISDN protocols if the I.451 recommendation did not place such a heavy bias on a non-symmetric (i.e. network side - user side) protocol specification. The models show that this bias will be both unnecessary and confusing for ISDN applications outside the local loop environment.

The enhancements proposed for the B-ISDN D channel to facilitate major simplifications to the data link layer for B-ISDN transmission links are important in several respects. Using out-of-slot bandwidth in the D channel to delimit the packet without resorting to header flags and 0-bit stuffing is certainly worth considering on its own merit. The concept of eliminating packet retransmission in layer 2 by applying FEC to the D channel at layer 1, while controversial, does lead one to ask if perhaps a less

complex layer 2 protocol could be used in B-ISDN. If packets rarely experience transmission errors, then perhaps a simplified layer 2 could be used with little or no impact on performance. With respect to the entire standards creation process, it shows that lower layer compatibility across all ISDN interfaces is not necessarily the optimal choice. For example, the technique of trading bandwidth used in the physical layer for simplifications in the higher layers has merit for bandwidth rich transmission media such as fiber.

Of all the standards related results, the simplifications made to the network layer's signalling protocol are the most dramatic. Decoupling the receive and transmit directions by using the two-way independent channel selection technique is an elegant solution to the normally messy problem of allocating bearer-channels in a symmetric protocol. The deviation from the I.451 protocol is quite minimal - more a matter of interpretation than of introducing new procedures. Further, this simplification is gained without reducing the functional capabilities of the signalling protocol. From a purely academic viewpoint, the simplification is well justified, but before the various standards committees would be willing to incorporate this type of signalling in the I.451 protocol it would have to be shown that the complexity of a table driven or firmware based implementation of the network layer processor would be sufficiently reduced to justify the loss of a flexible channel negotiation option. The purist might say it is always worth reducing complexity when there is no significant penalty to the functionality, but the realities of the process of adopting telecommunication standards might make two-way channel selection difficult to introduce at this late a stage.

6.3. Feasibility of a Hardware Based Protocol Processor

The concept of implementing layers 1 and 2 in hardware is not new, but implementing layer 3 in hardware was certainly a novel approach. Probably the most interesting aspect of trying to implement the network layer in hardware is the fundamental question of whether or not there is a basic technology mismatch between the call control procedures of I.451 and the capabilities of hardware based implementations. This research was only the first step in studying the network layer's suitability for hardware implementation, but still there are some general conclusions that can be drawn from the work.

The partitioning of the network layer into the PSM and ASM is an early recognition that some portions of the network layer must be tightly coupled to the specifics of the application's hardware. There was little work done on the specific requirements of the ASM with respect to various classes of applications, but one would expect that some of

the ASM's functionality must be common across most applications. An obvious example is the control of the time-slot interchange that decouples the transmission link from the internal bus in the two-way independent channel selection model. The functional requirements of the ASM is an area where much more work is needed before any general conclusions can be drawn concerning its suitability for hardware implementation.

The functionality required of the PSM was well covered by this research. There is no doubt that even our subset I.451 protocol is complex, but this complexity does not appear to exclude a hardware implementation based on the "system level integration" capabilities of modern VLSI circuits. With VLSI technology it is possible to design custom processors and peripherals and combine these with sufficient ROM and RAM to create a single chip computer that is specifically tuned to the processing requirements of the application. One could argue the semantics of whether this is a true "hardware" implementation, but the potential for a single chip solution for the ISDN signalling system's protocol processor is intriguing. Further, the software implementation of the PSM showed good "regularity", which leads one to conclude that even more complex versions of the network layer protocol could be implemented with the same basic software (or hardware) structure.

As a more general result, the nature of our proposed B-ISDN physical and data link layers resulted in a "stream processing" approach for the protocol processor's implementation. There is no reason to assume that because packet oriented data link layers are normally specified with a layer 2 - layer 3 interface based on the exchange of whole packets, it necessarily implies the actual implementation of the layer 2 - layer 3 interface cannot be based on a byte (i.e. stream) oriented interface. The reductions in buffering requirements and processing delay could be quite significant, and as long as the network layer is designed properly there should be no need to wait for the end of the packet to be received before packet processing can begin. The strength of this result can not be fully analyzed given the stage to which the research advanced. The reason for this is quite simple: the research showed that the requirement for packet and event buffering could be pushed all the way up to the ASM, but without having implemented a variety of ASM modules for specific applications it is impossible to determine if any "system level" complexity reduction accrues from a stream oriented layer 2 - layer 3 interface.

It should also be noted that although our initial layer 3 implementation is based on a stream oriented layer 2 - layer 3 interface, this type of interface is not a necessary component of a hardware implementation. If the standard LAPD protocol is used in the data link layer then the obvious implementation is to provide a memory sharing interface between layer 2 and layer 3. If this is the case, the implementation of the layer

3 protocol processor might as well take advantage of the fact that the entire packet is available at once. This would eliminate the need to process the signalling message in a strictly front-to-back manner, which may have some implementation advantages.

6.4. The Simulation System

The simulation system was designed to achieve three goals:

1) Test our subset network layer protocol for internal consistency and error handling capability.

2) Create a behavioral model of the protocol processor to study the functional partitioning and interfaces between modules.

3) Create a computing platform on which future demonstration systems could be built.

The simulation system successfully achieved its goals in all respects. As expected, the testing of the subset protocol proceeded smoothly: the CCITT has made considerable effort defining the original protocol, and the changes made to create the subset were well structured. By structuring the simulation software to mimic what was expected to occur in hardware, it was possible to study and test the behavior of the proposed implementation to a good level of detail. Finally, the effort spent in creating a user-friendly interface was well rewarded, and the benefits of this effort are expected to accrue should the simulation system be modified to test a different implementation or protocol.

6.5. Future Work

There is further work to be done with the simplified B-ISDN data link layer before it is ready for submission to the CCITT, but the potential for eliminating the processing complexity (and delay) of the current LAPD data link layer protocol is justification enough for further work in this area. Two-way independent channel selection is being presented at INFOCOM'88 in New Orleans on March 31.[18] The ATRC may also be presenting the concept to the ECSA and/or CCITT as a contribution to the ISDN standards.

The next stage of research in the network layer is to develop detailed ASM specifications for a reasonably complete set of applications. In parallel with this, the design of a VLSI based PSM should be initiated. For flexibility one would prefer that the specification of the signalling protocol be table driven. It would also be useful to have the state transition procedures table driven as well. The trick is to ensure that the chip count, complexity, processing speed and cost of such an implementation remains below what one would expect from a general purpose microprocessor based implementation. If it turns out that the fully flexible hardware solution is not cost effective then there are

really only two choices. Admit that the network layer is a software intensive process and accept the cost of a fully programmed implementation, or look for ways to reduce complexity by eliminating unnecessary flexibility. And of course the latter is is what we have been discussing all along!

References

1 Recommendation I.451, CCITT Red Book Volume III - Fascicle III.5, International Telecommunications Union, Geneva 1985.

2 Contribution T1D1.2/87-391 to ECSA.

3 Recommendation X.200, CCITT Red Book Volume VIII - Fascicle VIII.5, International Telecommunications Union, Geneva 1985.

4 Kano, S., "Layers 2 and 3 ISDN recommendations," *IEEE Journal on Selected Areas in Communications*, vol. sac-4, no. 3, pp 355-9, May 1986.

5 Chapter 6, Professional Program Sessional Record, Wescon/86 Electronic Show and Convention. November 18-20, 1986. Anaheim, California. (Sponsored by IEEE)

6 *Telesis*, Vol 13, No 3, 1986: p. 2 (Box 2). BNR, Ottawa.

7 *Micro Communications Handbook* (1988) #231658. Intel Literature Sales, Box 58130, Santa Clara, California. 95052-8130. pp. 5.92-5.97.

8 *Telecommunications Solutions* (1987) #09558A. Advanced Micro Devices, Box 3453, Sunnyvale, California. 94088.

9 Tanenbaum, Andrew S. *Computer Networks*. Englewood Cliffs, New Jersey: Prentice-Hall (1981).

10 Schwartz, Mischa. *Telecommunication Networks: Protocols, Modeling and Analysis*. Don Mills, Ontario: Addison-Wesley Publishing (1987).

11 In North America the best source of CCITT information is Omnicom Inc. Suite 206 - 501 Church St. N.E., Vienna, Virgina. 22180

12 ECSA, 5420 Grosvenor Lane, Bethesda, Maryland. 20814-2122.

13 Bartee, Thomas C. (editor). *Digital Communications*. Indianapolis, Indiana: Howard W. Sams & Co. (1986), Chapter 3.

14 Miller, Michael J. and Ahamed, Syed V. *Digital Transmission Systems and Networks* (V. II). Rockville, Maryland: Computer Science Press (1988). Ch 5.

15 Fletcher, William I. *An Engineering Approach to Digital Design*. Englewood Cliffs, New Jersey: Prentice-Hall (1980), pp. 480-485.

16 Brown, Jeffrey J. *Addendum to MSc Thesis (1988)*. ATRC, #200, 4245-97 Street, Edmonton, Alberta, T6E 5Y4.

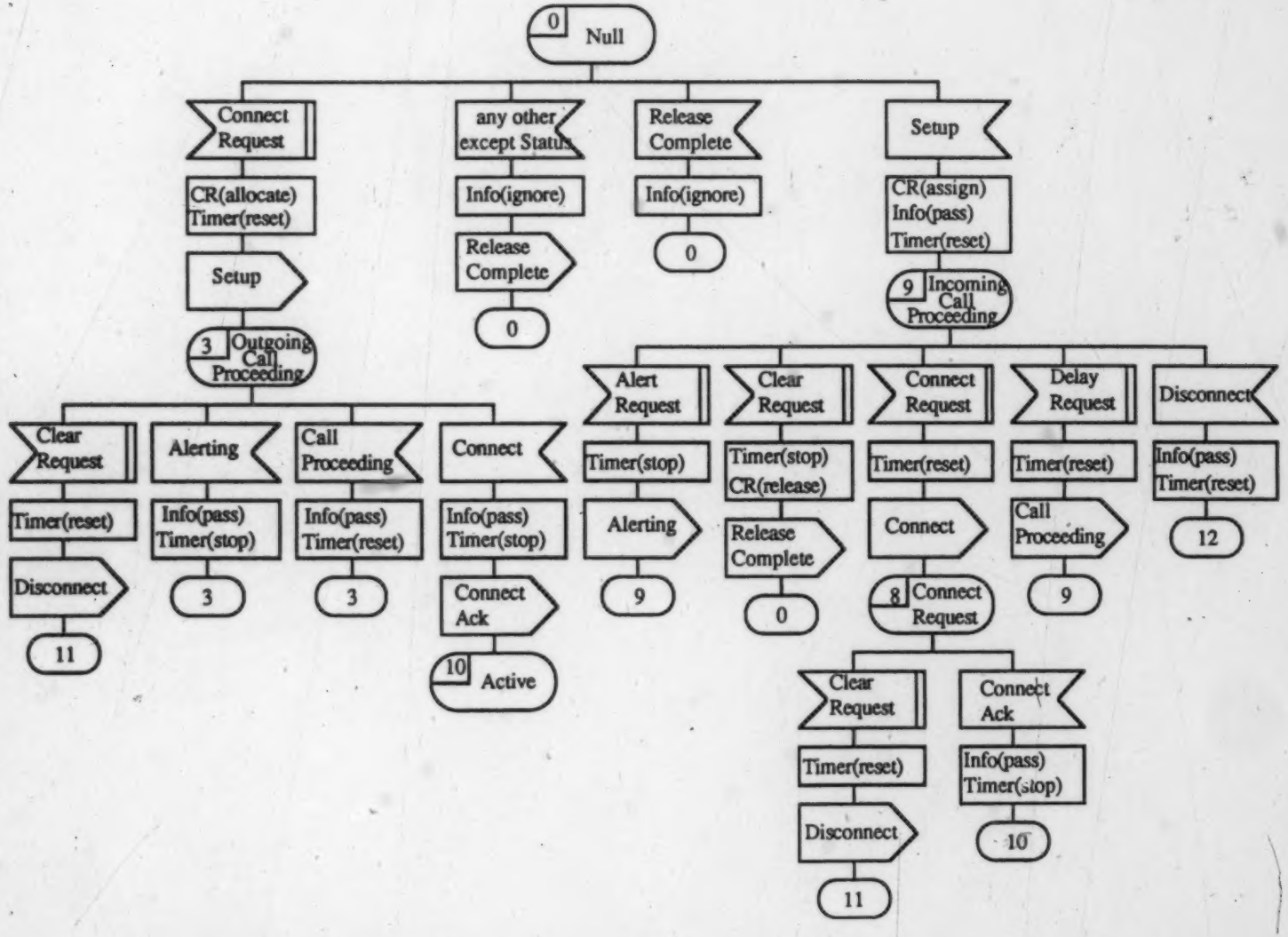
17 Brooks, F.P Jr. "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, Vol. 20, No.4, April 1987, pp. 10-19

18 Brown, Jeffrey J., Krzymien, Witold A. "A Subset of the ISDN Network Layer Protocol Suitable for Implementation in Hardware", INFOCOM'88 Conference Proceedings (in print).

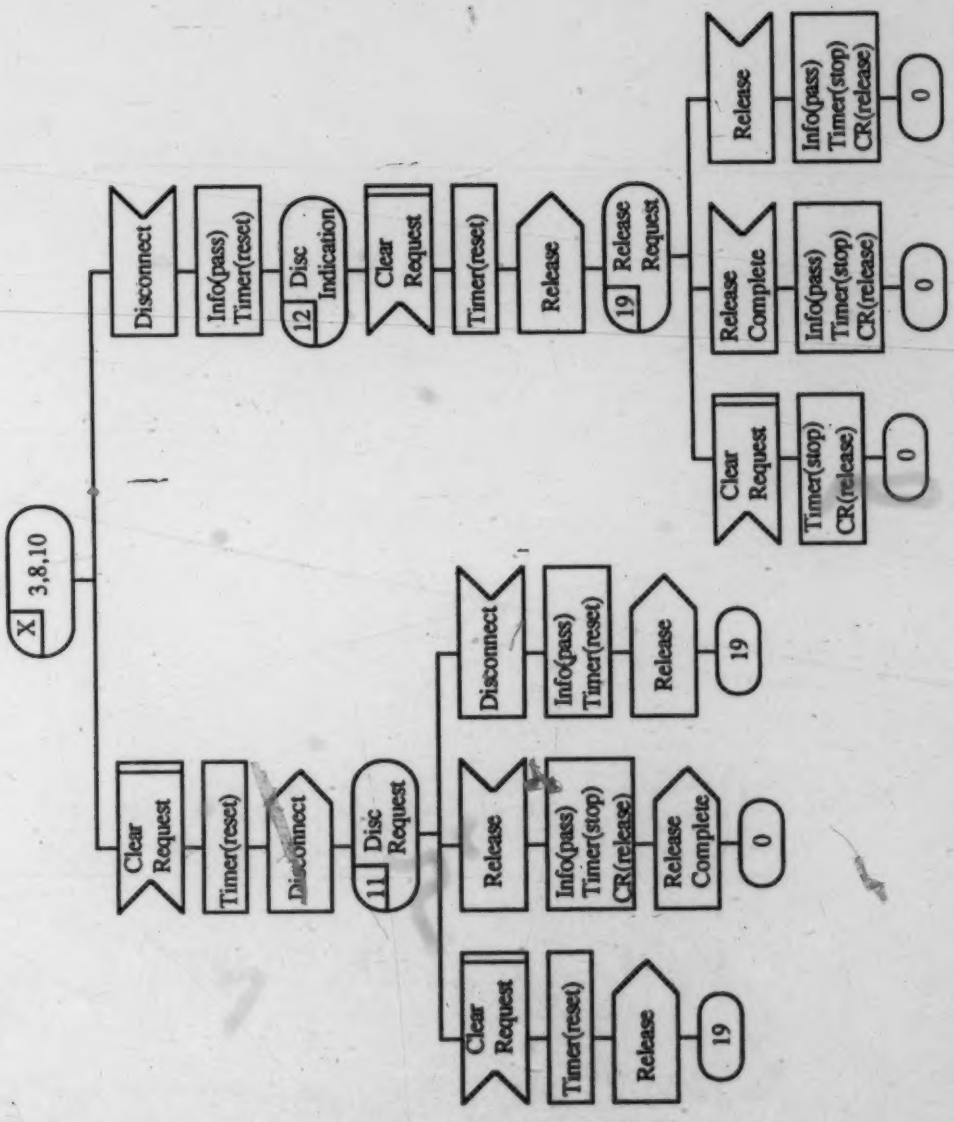
Appendix A

Contents

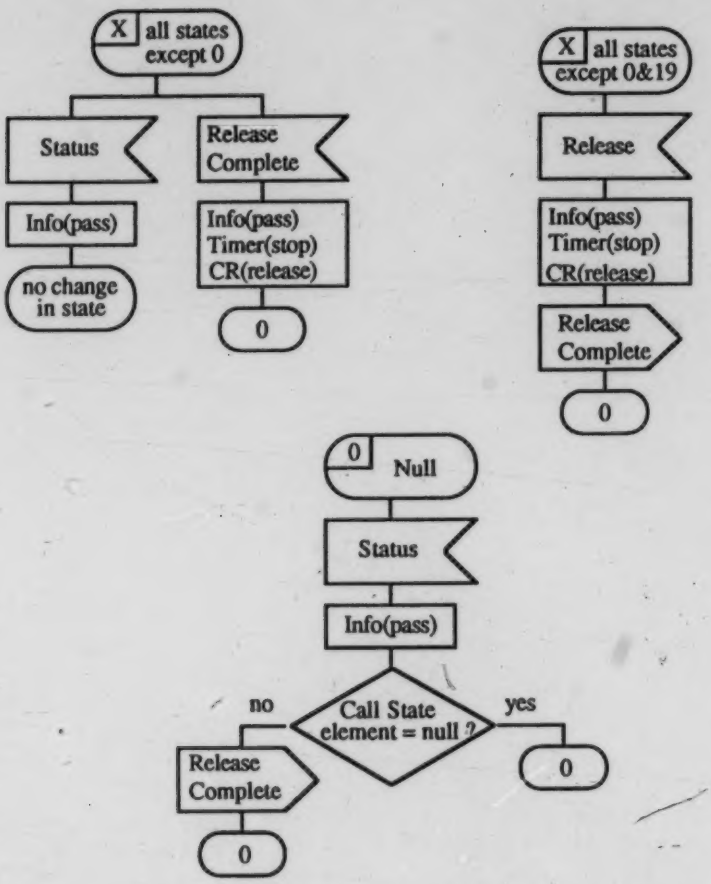
Subset I.451 SDL Diagrams
Subset I.451 Message Content Diagrams
Subset I.451 Message Content Table



Subset I.451 SDL Diagrams (part 1 of 3)



Subset I.451 SDL Diagrams (part 2 of 3)

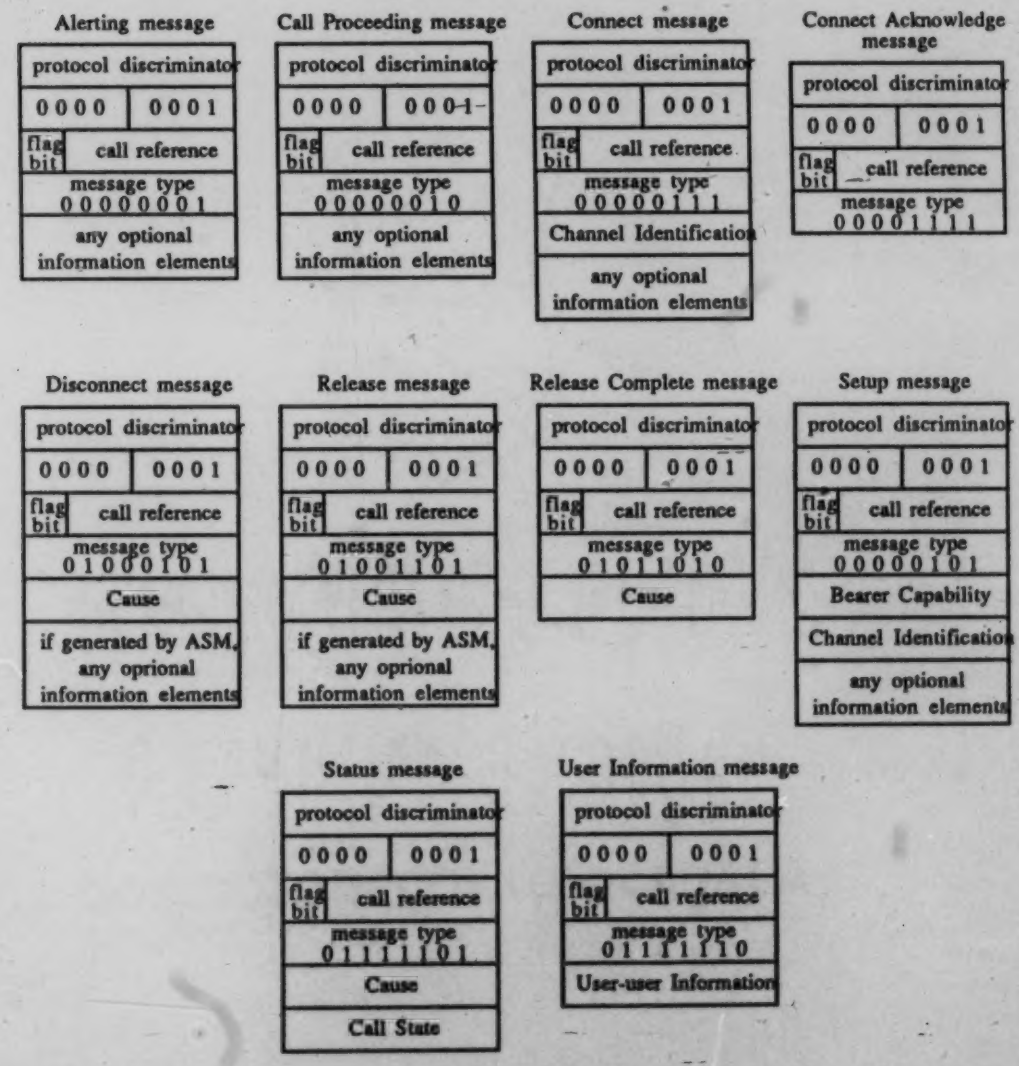


Legend

CR(allocate)	- PSM allocates a new Call Reference for the call.
CR(assign)	- PSM allocates the Call Reference specified by the Setup message.
CR(free)	- PSM frees the Call Reference.
Timer(reset)	- PSM starts or resets timer assigned to this Call Reference.
Timer(stop)	- PSM stops timer for this Call Reference.
Info(ignore)	- PSM does not notify ASM that message has been received.
Info(pass)	- PSM notifies ASM that message has been received and passes the ASM any valid information elements contained in the message.
Info(notify)	- PSM notifies ASM that a forced jump to state 19 has been made.

Subset I.451 SDL Diagrams (part 3 of 3)

The following diagrams show the contents of the valid messages for the subset I.451 protocol. All mandatory information elements are listed below the call reference field. If optional information elements are permitted, this is also shown. Note that the subset protocol does not permit the Channel Identification information element to appear in any messages other than Setup and Connect. The list of valid optional elements for those messages that have them can be found in the I.451 recommendation.



Subset I.451 Message Content Diagrams

The table defines which messages are permitted in the subset Q.931 protocol, and delineates any differences in message contents. "Source" specifies whether the information elements are generated by the ASM or the PSM.

Message	Source	Differences from I.451	Notes
Setup	ASM	Channel Identification is mandatory.	Channel Identification is calling-to-called party direction only.
Alerting	ASM	Channel Identification is not allowed.	
Call Proceeding	ASM	Channel Identification is not allowed.	
Connect	ASM	Channel Identification is mandatory.	Channel Identification is called-to-calling party direction only.
Connect Acknowledge	PSM	No optional information elements permitted.	
Disconnect	ASM	No differences.	Normal call disestablishment.
	PSM	No optional information elements permitted.	Disconnect sent as a result of timeout. Cause value is 111.
Release	ASM	No differences.	Normal call disestablishment.
	PSM	No optional information elements permitted.	Release sent as a result of timeout. Cause value is 111.
Release Complete	PSM	No optional information elements permitted.	
Status	PSM	No optional information elements permitted.	
User Info	ASM	No differences.	

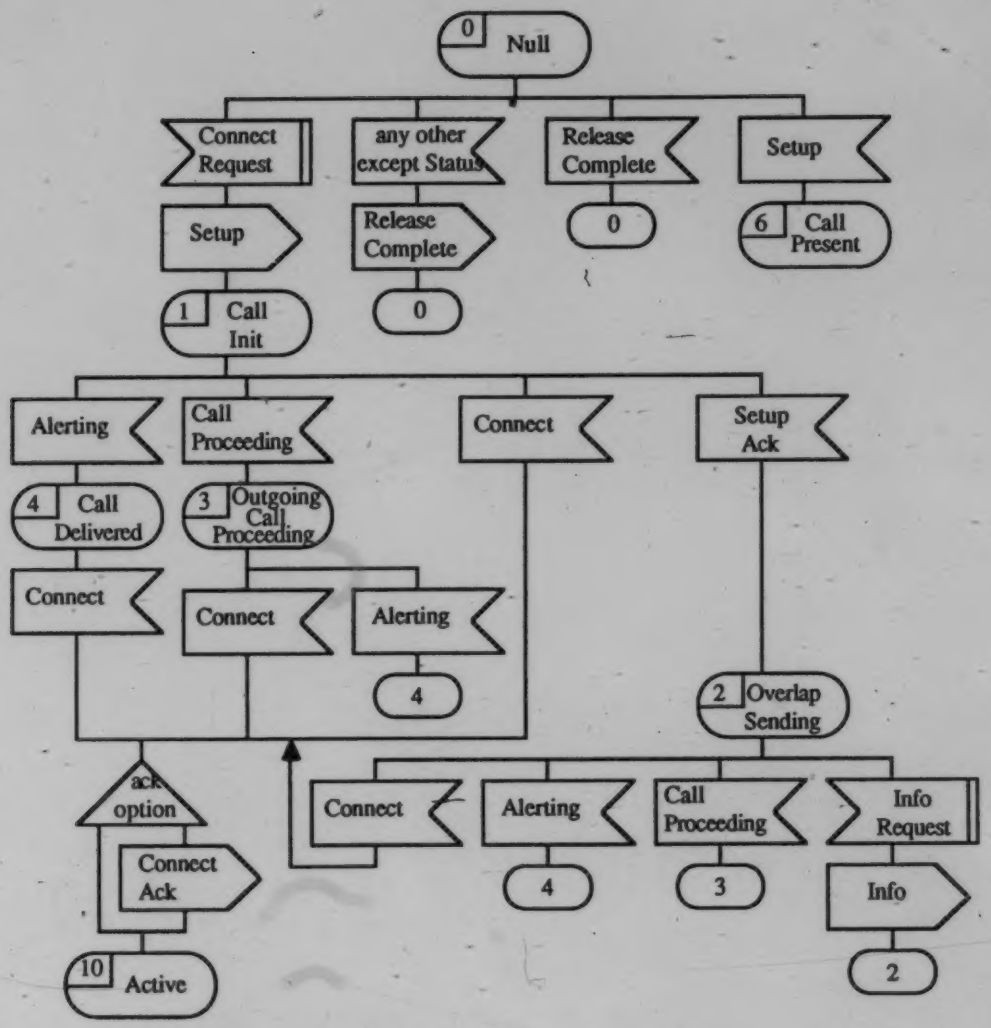
Subset I.451 Message Content Table

Appendix B

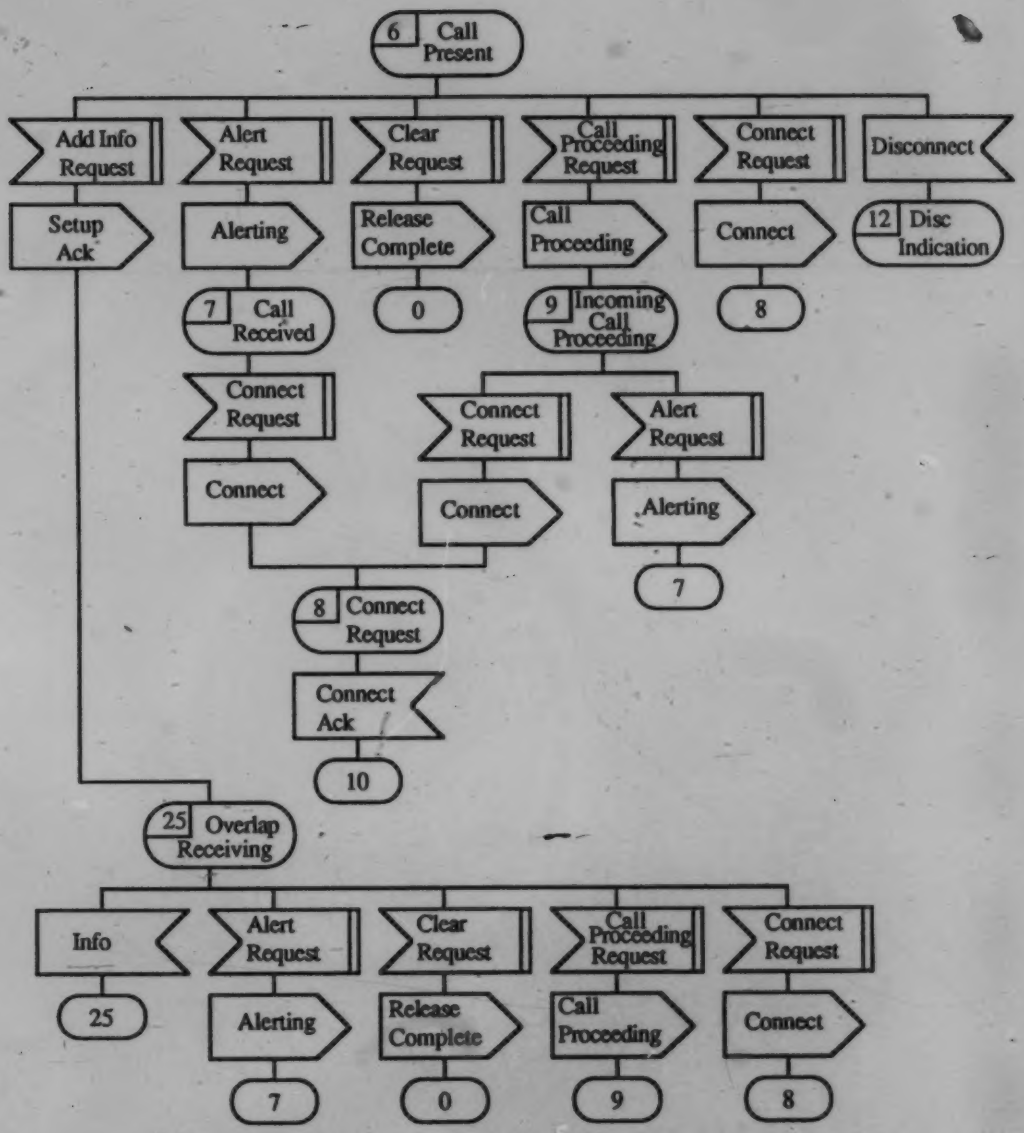
Contents

Original I.451 SDL Diagrams

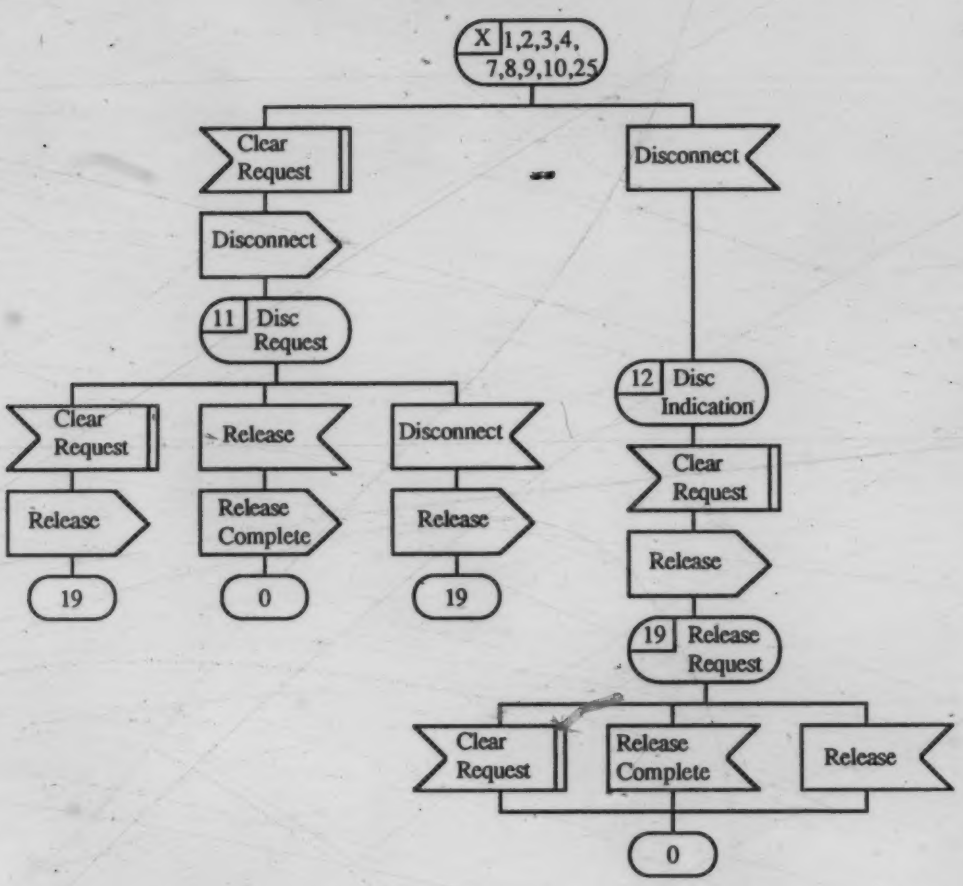
The diagrams in this appendix are derived from the August 1987 version of I.451. The timeout procedures have been excluded, as have the Suspend and Resume procedures. Error handling procedures are the same as for the subset protocol. Other non-essential details have also been left out for clarity.



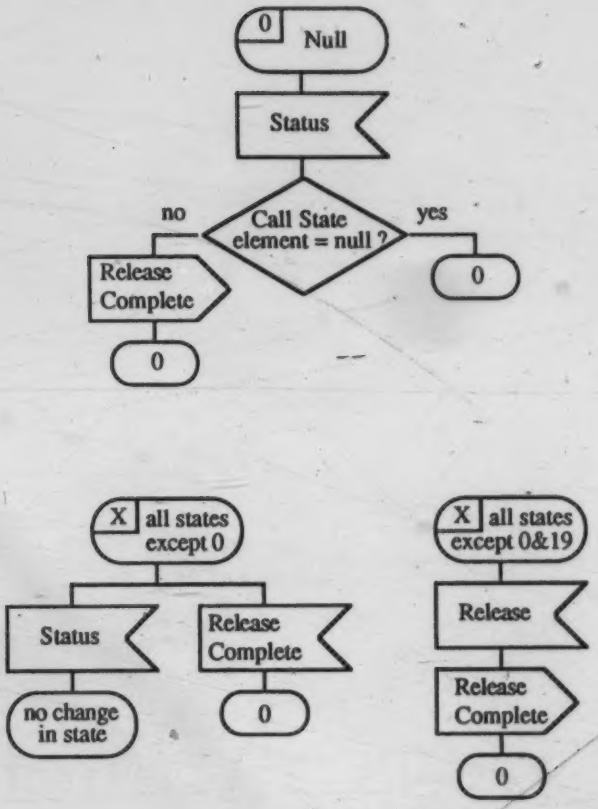
Original I.451 SDL Diagrams (part 1 of 4)



Original I.451 SDL Diagrams (part 2 of 4)



Original I.451 SDL Diagrams (part 3 of 4)



Original I.451 SDL Diagrams (part 4 of 4)

Appendix C

Recommendation	Area covered
X.210	Definition conventions for describing the services (i.e. functionality) of the layers of the OSI Reference Model.
X.213	The general functionality of the network layer within the OSI Reference Model.
I.130	General modeling methods for the characterization of telecommunication services and network capabilities in an ISDN. Here is where the concept of service attributes is introduced and defined.
I.210	A method of describing services offered by an ISDN. This defines the concepts of bearer services (and hence bearer capabilities) and teleservices.
I.211	Recommended bearer services to be supported by an ISDN.
I.212	Recommended teleservices to be supported by an ISDN.
I.320	An extension of the X.200 series of recommendations to create a protocol model that is specific to ISDN telecommunication networks.
I.330	General concepts of network addressing for ISDN.
I.331	The recommended world wide ISDN numbering plan to integrate the various voice, packet data, etc. numbering plans that currently exist.
I.340	A definition of the connection types necessary to create the bearer services and teleservices defined in the I.200 series of recommendations.
I.412	The definition of the interface structures, channel types, etc. for ISDN user-network physical interfaces. Here is where the B, H and D channels are defined.
I.450	The specific functionality required of the D channel network layer in an ISDN user-network interface. Although this is specific to user-network interfaces, many of the functions apply to all types of interfaces that support switched services. (e.g.. a PBX-terminal interface or a LAN-terminal interface)

ISDN Related Recommendations

Appendix D

Simulation code for the EPM Module

```
IMPLEMENTATION OF EPM;

uses  messageConstants,fifoInterface,EPMInterface;

{ ******************** local declarations ********************* }

const
    CALLREFTABLESIZE = 255;
    CALLREFMASK      = 2#10000000;

type
    CALLREFTABLERANGE = WRD(0)..CALLREFTABLESIZE;
    CALLREFTABLETYPE =
      record
          callState     : VALIDSTATETYPE;
          ticsLeft      : BYTE;
          timedOut      : BOOLEAN;
      end;
    EVENTTYPE             = (NOEVENT,INCOMINGMSG,INTERNALREQUEST,TIMEOUT);
    MA                    = array[VALIDMSGTYPE] of BOOLEAN;
    VALIDMSGARRAY         = ARRAY[VALIDSTATETYPE] of MA;

const
    validMsg = VALIDMSGARRAY (
  { ALERT,CPROC,CONN  ,CACK  ,DISC  ,REL  ,RELC,SETUP,STAT,UINFO}
MA(TRUE  ,TRUE  ,TRUE  ,TRUE  ,TRUE  ,TRUE,TRUE,TRUE  ,TRUE,TRUE  ),{NULL}
MA(TRUE  ,TRUE  ,TRUE  ,FALSE,TRUE  ,TRUE,TRUE,FALSE,TRUE,FALSE),{OUTCALL}
MA(FALSE,FALSE,FALSE,FALSE,TRUE  ,TRUE,TRUE,FALSE,TRUE,FALSE),{INCALL}
MA(FALSE,FALSE,FALSE,TRUE  ,TRUE  ,TRUE,TRUE,FALSE,TRUE,FALSE),{CONNREQ}
MA(FALSE,FALSE,FALSE,FALSE,TRUE  ,TRUE,TRUE,FALSE,TRUE,TRUE  ),{ACTIVE}
MA(FALSE,FALSE,FALSE,FALSE,TRUE  ,TRUE,TRUE,FALSE,TRUE,FALSE),{DISCREQ}
MA(FALSE,FALSE,FALSE,FALSE,FALSE,TRUE,TRUE,FALSE,TRUE,FALSE),{DISCIND}
MA(FALSE,FALSE,FALSE,FALSE,FALSE,TRUE,TRUE,FALSE,TRUE,FALSE)  {RELREQ}
);

var [public]   {public for debugging}
{universal variables for all processing}
    callRefTable          : array [CALLREFTABLERANGE] of CALLREFTABLETYPE;
    activeCallRef         : CALLREFTYPE;
    activeCallState       : VALIDSTATETYPE;
    eventSelectEnable     : BOOLEAN;
    activeEvent           : EVENTTYPE;

{output line buffering variables for fifo's and ASM interfaces}
    rxFifoPopDataBuf        : BOOLEAN;
    rxFifoResetBuf          : BOOLEAN;
    txFifoOutputEnableBuf   : BOOLEAN;
    txFifoResetBuf          : BOOLEAN;
    txFifoPushBuf           : BOOLEAN;
    txFifoMsgBuf            : VALIDMSGTYPE;
    txFifoCauseBuf          : BYTE;
    txFifoDiagBuf           : BYTE;
    txFifoCallRefBuf        : BYTE;
    txFifoElToFollowBuf     : BOOLEAN;
    txFifoCallStateBuf      : BYTE;
    EPMReqCtrlLineBuf       : EPMREQLINETYPE;
    EPMTOutLineBuf          : EPMTOUTLINETYPE;
```

```
    EPMMsgLineBuf           : EPMMSGLINETYPE;
    EPMElmLineBuf           : EPMELMLINETYPE;
    EPMMsgOctetBuf          : BYTE;
    EPMMsgTypeBuf           : VALIDMSGTYPE;
    EPMElementTypeBuf       : VALIDELEMENTTYPE;
    EPMCallRefOutBuf        : BYTE;
    EPMStateBuf             : VALIDSTATETYPE;

(timeout processing variables)
    timeoutPending      : BOOLEAN;
    timeoutCallRef      : CALLREFTYPE;

(internal request processing variables)
    requestState : (NEWREQ,GETTINGREQ,HANDOFFTOASM);

(incoming message processing variables)
    msgState            : (MSGPD,MSGCRLEN,MSGCALLREF,MSGMSGTYPE,
                           MSGPROCESS,MSGSKIP);
    processNextOctet    : BOOLEAN;
    activeMsg           : VALIDMSGTYPE;
    activeElement       : VALIDELEMENTTYPE;
    mandatoryElements   : set of VALIDELEMENTTYPE;
    allowedElements     : set of VALIDELEMENTTYPE;
    elementState        : (ELSTART,ELGETLEN,ELGETLENTHENSKIP,ELSKIP,ELPASS);
    elementSkipLength   : BYTE;
    releaseCause        : BYTE;
    relCauseState       : (RCSNONE,RCSOCTET3,RCSOCTET3A,RCSOCTET4);
    statusState         : BYTE;
    statStateState      : (SSSNONE,SSSOCTET3);

{ ********************* local pascal routines ********************** }

procedure InitEPM;

var
    cr : CALLREFTYPE;

begin
for cr:= 0 to CALLREFTABLESIZE do with callRefTable[cr] do
    begin
    callState     := NULLSTATE;
    ticsLeft      := 0;
    timedOut      := FALSE;
    end;
timeoutPending          := FALSE;
eventSelectEnable       := TRUE;
rxFifoPopDataBuf        := FALSE;
rxFifoResetBuf          := FALSE;
txFifoOutputEnableBuf   := TRUE;
txFifoResetBuf          := FALSE;
txFifoPushBuf           := FALSE;
EPMReqCtrlLineBuf       := RLNOREQ;
EPMTOutLineBuf          := TLNOTOUT;
EPMMsgLineBuf           := MLNOMSG;
EPMElmLineBuf           := ELNOELM;
end;

{ ********************************************** }

procedure UpdateTimers;

var
    cr : CALLREFTYPE;
```

```
begin
timeoutPending := FALSE;
for cr:= 1 to CALLREFTABLESIZE do with callRefTable[cr] do
   begin
   if ticsLeft <> 0 then
      begin
      ticsLeft := ticsLeft - 1;
      if ticsLeft = 0 then
         timedOut := TRUE;
      end;
   if timedOut then
      begin
      timeoutPending := TRUE;
      timeoutCallRef := cr;
      end;
   end;
end;

{ ********************************************* }

procedure StopTimer;

begin
with callRefTable[activeCallRef] do
   begin
   ticsLeft := 0;
   timedOut := FALSE;
   end;
end;

{ ********************************************* }

procedure ResetTimer;

begin
with callRefTable[activeCallRef] do
   begin
   ticsLeft := EPMTimeoutValue;
   timedOut := FALSE;
   end;
end;

{ ********************************************* }

procedure MsgPrep;

begin
activeEvent       := INCOMINGMSG;
msgState          := MSGPD;
end;

{ ********************************************* }

procedure SelectActiveEvent;

begin
if rxFifoFull then
   MsgPrep
else if NOT txFifoEmpty then
   begin
   if rxFifoDataReady then
      MsgPrep
   else
      activeEvent := NOEVENT;
```

```
    end
else  (for this else clause the tx fifo is empty)
    begin
     if EPMEventRequest = REQPENDING then
        begin
         activeEvent := INTERNALREQUEST;
         requestState := NEWREQ;
        end
     else if rxFifoDataReady then
        MsgPrep
     else if timeoutPending and (EPMEventRequest = NOEVENTREQ) then
         activeEvent := TIMEOUT
    else
         activeEvent := NOEVENT;
    end;
end;

{ ********************************************** }

procedure NewCallState( state : VALIDSTATETYPE );

begin
callRefTable[activeCallRef].callState := state;
activeCallState := state;
end;

{ ******************************************************** }
{ ******************************************************** }

procedure MsgProcessing;

{ ********* nested routines ********** }

procedure SendStatusMsg( cause,diag : BYTE );
begin
txFifoPushBuf          := TRUE;
txFifoMsgBuf           := STATUS;
txFifoCauseBuf         := cause;
txFifoDiagBuf          := diag;
txFifoCallStateBuf     := stateCodes[activeCallState];
txFifoCallRefBuf       := activeCallRef;
txFifoElToFollowBuf    := FALSE;
end;

procedure SendConnAckMsg;
begin
txFifoPushBuf          := TRUE;
txFifoMsgBuf           := CONNACK;
txFifoCallRefBuf       := activeCallRef;

{The following are dummy assignments}
txFifoCauseBuf         := 0;
txFifoDiagBuf          := 0;
txFifoCallStateBuf     := 0;
end;

procedure SendRelMsg( cause : BYTE );
begin
txFifoPushBuf          := TRUE;
txFifoMsgBuf           := RELEASE;
txFifoCallRefBuf       := activeCallRef;
txFifoCauseBuf         := cause;
txFifoElToFollowBuf    := FALSE;
```

```
(The following are dummy assignments)
txFifoDiagBuf          := 0;
txFifoCallStateBuf     := 0;
end;

procedure SendRelCompMsg( cause : BYTE );
begin
txFifoPushBuf          := TRUE;
txFifoMsgBuf           := RELCOMP;
txFifoCallRefBuf       := activeCallRef;
txFifoCauseBuf         := cause;
txFifoElToFollowBuf    := FALSE;

(The following are dummy assignments)
txFifoDiagBuf          := 0;
txFifoCallStateBuf     := 0;
end;

function AssignActiveMsg : BOOLEAN;
var mt : VALIDMSGTYPE;
begin
AssignActiveMsg := FALSE;
for mt := ALERT to USERINFO do
    if rxFifoData = msgCodes[mt] then
       begin
       activeMsg := mt;
       AssignActiveMsg := TRUE;
       end;
end;

function AssignActiveElement : BOOLEAN;
var et   : VALIDELEMENTTYPE;
begin
AssignActiveElement := FALSE;
for et := BCAP to UUINFO do
    if rxFifoData = elementCodes[et] then
       begin
       activeElement := et;
       AssignActiveElement := activeElement in allowedElements;
       end;
end;

procedure PassMsgInfoToASM( ctrlLine : EPMMSGLINETYPE);
begin
EPMMsgLineBuf      := ctrlLine;
EPMMsgTypeBuf      := activeMsg;
EPMCallRefOutBuf   := activeCallRef;
EPMStateBuf        := activeCallState;
end;

procedure PassElementInfoToASM( ctrlLine : EPMELMLINETYPE );
begin
EPMElmLineBuf      := ctrlLine;
EPMElementTypeBuf  := activeElement;
EPMMsgOctetBuf     := rxFifoData;
end;

procedure SaveRelCause;
begin
case relCauseState of
   RCSNONE:
      begin
      { should never occur }
      end;
```

```
    RCSOCTET3:
       begin
        if (rxFifoData AND 2#10000000) = 0 then
           relCauseState := RCSOCTET3A
       else
           relCauseState := RCSOCTET4;
       end;
    RCSOCTET3A:
       begin
        if (rxFifoData AND 2#10000000) = 0 then
           relCauseState := RCSOCTET3A
       else
           relCauseState := RCSOCTET4;
       end;
    RCSOCTET4:
       begin
        releaseCause := rxFifoData;
        relCauseState := RCSNONE;
       end;
   end; (case)
end;

procedure SaveStatState;
begin
statusState := rxFifoData;
statStateState := SSSNONE;
end;

procedure ProcessMsg;
begin
case elementState of
    ELSTART:
       begin
        if AssignActiveElement then
          begin
           PassElementInfoToASM( ELSOELM );
           elementState := ELGETLEN;
           if ((activeMsg = DISC) or (activeMsg = RELEASE)) and
               (activeElement = CAUSE) then
              relCauseState := RCSOCTET3
           else if (activeMsg = STATUS) and
                   (activeElement = CALLSTATE) then
              statStateState := SSSOCTET3
           else
              begin
               statStateState := SSSNONE;
               relCauseState := RCSNONE;
              end
          end
       else
          begin
           SendStatusMsg( 99, rxFifoData );
           elementState := ELGETLENTHENSKIP;
          end;
       end;
    ELGETLEN:
       begin
        elementSkipLength := rxFifoData;
        if elementSkipLength <> 0 then
          begin
           PassElementInfoToASM( ELELM );
           elementState := ELPASS;
          end
       else
```

```
            begin
             PassElementInfoToASM( ELEOELM );
             SendStatusMsg( 100, elementCodes[activeElement] );
            elementState := ELSTART;
            end;
         end;
      ELGETLENTHENSKIP:
         begin
          elementSkipLength := rxFifoData;
          if elementSkipLength <> 0 then
             elementState := ELSKIP
         else
             elementState := ELSTART;
         end;
      ELSKIP:
         begin
          elementSkipLength := elementSkipLength - 1;
          if elementSkipLength = 0 then
             elementState := ELSTART;
         end;
      ELPASS:
         begin
          elementSkipLength := elementSkipLength - 1;
          if elementSkipLength = 0 then
            begin
             mandatoryElements := mandatoryElements - [activeElement];
             PassElementInfoToASM( ELEOELM );
             elementState := ELSTART;
            end
         else
             PassElementInfoToASM( ELELM );
          if relCauseState <> RCSNONE then
             SaveRelCause;
          if statStateState <> SSSNONE then
             SaveStatState;
         end;
      end; (case)
end;

function FindMissingElement : BYTE;
var   et : VALIDELEMENTTYPE;
begin
for et := BCAP to UUINFO do
    if et in mandatoryElements then
       FindMissingElement := elementCodes[et];
end;

procedure ProcessEOM;
begin
if elementState <> ELSTART then
   begin
    PassElementInfoToASM( ELEOELMBAD );
    SendStatusMsg( 100, elementCodes[activeElement] );
   end;
if mandatoryElements <> [] then
   begin
    PassMsgInfoToASM( MLEOMSGBAD );
    if activeMsg = SETUP then
       SendRelCompMsg(96)
   else
       SendStatusMsg( 96, FindMissingElement );
   end
else
   begin
```

```
case activeMsg of
   ALERT:
       begin  (state will be outgoing call proc)
      StopTimer;
      end;
   CALLPROC:
       begin  (state will be outgoing call proc)
      ResetTimer;
      end;
   CONNECT:
       begin  (state will be outgoing call proc)
      StopTimer;
      SendConnAckMsg;
       NewCallState( ACTIVESTATE );
      end;
   CONNACK:
       begin  (state will be connect request)
      StopTimer;
       NewCallState( ACTIVESTATE );
      end;
   DISC:
      begin
       if activeCallState = DISCREQSTATE then
         begin
          SendRelMsg( releaseCause );
         ResetTimer;
          NewCallState( RELREQSTATE);
         end
       else     (activeCallState = OUTCALLPROCSTATE  or
                 activeCallState = INCALLPROCSTATE  or
                 activeCallState = CONNREQSTATE  or
                 activeCallState = ACTIVESTATE )
         begin
         ResetTimer;
          NewCallState( DISCINDICATIONSTATE );
         end
      end;
   RELEASE:
      begin
       if activeCallState <> RELREQSTATE then
          SendRelCompMsg( releaseCause );
      StopTimer;
       NewCallState( NULLSTATE );
      end;
   RELCOMP:
      begin
      StopTimer;
       NewCallState( NULLSTATE );
      end;
   SETUP:  (state will be null)
      begin
      ResetTimer;
       NewCallState( INCALLPROCSTATE );
      end;
   STATUS:
      begin
       if (activeCallState = NULLSTATE) and
          (statusState <> stateCodes[NULLSTATE]) then
         SendRelCompMsg( 101 );
      end;
   USERINFO:  (state will be active)
      begin
      end;
   end; (case)
```

```
    PassMsgInfoToASM( MLEOMSGGOOD );
   end;
end;

procedure PrepareForMsgProcessing;
begin
PassMsgInfoToASM( MLSOMSG );
elementState := ELSTART;
releaseCause := 96;
case activeMsg of
   ALERT:
      begin
       mandatoryElements := [];
       allowedElements   := [UUINFO];
      end;
   CALLPROC:
      begin
       mandatoryElements := [];
       allowedElements   := [];
      end;
   CONNECT:
      begin
       mandatoryElements := [CHID];
       allowedElements   := [CHID,UUINFO];
      end;
   CONNACK:
      begin
       mandatoryElements := [];
       allowedElements   := [];
      end;
   DISC:
      begin
       mandatoryElements := []; {This is tricky, but it handles
                                 a missing cause info element
                                 nicely since releaseCause is
                                 initialized to 96. }
       allowedElements   := [CAUSE,UUINFO];
      end;
   RELEASE:
      begin
       mandatoryElements := [];  (see disc note)
       allowedElements   := [CAUSE,UUINFO];
      end;
   RELCOMP:
      begin
       mandatoryElements := [];  (see disc note)
       allowedElements   := [CAUSE];
      end;
   SETUP:
      begin
       mandatoryElements := [BCAP,CHID];
       allowedElements   := [BCAP,CALLEDNUM,CALLINGNUM,CHID,UUINFO];
      end;
   STATUS:
      begin
       mandatoryElements := [CAUSE,CALLSTATE];
       allowedElements   := [CAUSE,CALLSTATE];
      end;
   USERINFO:
      begin
       mandatoryElements := [UUINFO];
       allowedElements   := [UUINFO];
      end;
   end; (case)
```

```
msgState := MSGPROCESS;
if rxFifoDataType = RXFIFOEOM then
    ProcessEOM;
end;

{ ********* MsgProcessing ********** }

begin
case msgState of
   MSGPD:
      begin
       if (rxFifoDataType=RXFIFOSOM) and (rxFifoData=DISCRIMINATOR) then
          msgState := MSGCRLEN
      else
          msgState := MSGSKIP;
      end;
   MSGCRLEN:
      begin
       if (rxFifoDataType=RXFIFOMSG) and (rxFifoData=CALLREFLEN) then
          msgState := MSGCALLREF
      else
          msgState := MSGSKIP;
      end;
   MSGCALLREF:
      begin
       if rxFifoDataType = RXFIFOMSG then
         begin
          { At this point the call reference must be "flipped" since
            the far side's view of called party and calling party is
            the reverse of this end's view. }
          if (rxFifoData AND CALLREFMASK) = 0 then  {if calling party}
             activeCallRef :=
                 rxFifoData OR CALLREFMASK  {set bit 7: called party}
          else
             activeCallRef :=
                 rxFifoData AND (NOT CALLREFMASK);  {clear bit 7}
          activeCallState := callRefTable[activeCallRef].callState;
          msgState := MSGMSGTYPE;
         end
      else
          msgState := MSGSKIP;
      end;
   MSGMSGTYPE:
      begin
       if    (rxFifoDataType=RXFIFOMSG) or
             (rxFifoDataType=RXFIFOEOM) then
         begin
          if not AssignActiveMsg then
             begin
              SendStatusMsg( 97, rxFifoData );
              msgState := MSGSKIP;
             end
          else
             begin
              if ((activeCallRef=0) or (activeCallRef=CALLREFMASK))
                    and (activeMsg <> STATUS) then
                 begin
                  SendStatusMsg( 81, activeCallRef );  {invalid callref}
                  msgState := MSGSKIP;
                 end
              else if activeMsg = SETUP then
                 begin
                  if (activeCallRef AND CALLREFMASK)=0 then
                     msgState := MSGSKIP  {remember flag is flipped}
```

```
                else if activeCallState = NULLSTATE then
                   PrepareForMsgProcessing
                else
                   msgState := MSGSKIP;
                end
              else if NOT ValidMsg[activeCallState,activeMsg] then
                begin
                 SendStatusMsg( 101, rxFifoData );
                 msgState := MSGSKIP;
                end
              else if activeCallState = NULLSTATE then
                begin
                 if activeMsg = RELCOMP then
                    msgState := MSGSKIP
                 else if activeMsg = STATUS then
                    PrepareForMsgProcessing
                 else
                    begin
                     SendRelCompMsg( 81 );
                     msgState := MSGSKIP;
                    end;
                end
              else
                  PrepareForMsgProcessing;
              end
            end;
          end;
        MSGPROCESS:
          begin
           if (rxFifoDataType = RXFIFOERROR) or
              (rxFifoDataType = RXFIFOSOM) then  (actually it can't be SOM)
              PassMsgInfoToASM( MLEOMSGBAD )
          else
             begin
              ProcessMsg;
              if rxFifoDataType = RXFIFOEOM then
                 ProcessEOM;
             end;
          end;
        MSGSKIP:
          begin
          end;
        end; (case)
      RxFifoPopDataBuf := TRUE;
      eventSelectEnable := (rxFifoDataType = RXFIFOERROR) or
                           (rxFifoDataType = RXFIFOEOM);
      end;

      { ********************************************************* }
      { ********************************************************* }

      procedure RequestProcessing;

      { ********* nested routines ********** }

      procedure PassReqInfoToASM( ctrlLine : EPMREQLINETYPE );
      begin
      EPMReqCtrlLineBuf := ctrlLine;
      EPMCallRefOutBuf  := activeCallRef;
      EPMStateBuf       := activeCallState;
      end;

      procedure SetupForRequest;
      begin
```

```
activeCallRef    := EPMCallRefIn;
activeCallState  := callRefTable[activeCallRef].callState;
end;

function GetFreeCallRef:CALLREFTYPE;
var
   i : CALLREFTYPE;
begin
i := 1;
while i <> 0 do with callRefTable[i] do
   if callState = NULLSTATE then
      begin
       GetFreeCallRef := i;
      i := 0;
      end
   else
      begin
       if i = NOT CALLREFMASK then  {loop for 1..127}
         begin
          GetFreeCallRef := 0;
         i := 0;
         end
      else
         i := i+1;
      end;
end;

procedure SendMsg( mt:VALIDMSGTYPE; moreEl:BOOLEAN );
begin
txFifoPushBuf          := TRUE;
txFifoMsgBuf           := mt;
txFifoCallRefBuf       := activeCallRef;
txFifoElToFollowBuf    := moreEl;

{The following are dummy assignments}
txFifoCauseBuf      := 0;
txFifoDiagBuf       := 0;
txFifoCallStateBuf  := 0;
end;

{ ********* RequestProcessing ********** }

begin
if requestState = NEWREQ then
   begin
   PassReqInfoToASM( RLREQACK );
   requestState := GETTINGREQ;
   end
else if requestState = GETTINGREQ then
   begin
   case EPMEventRequest of
      NOEVENTREQ,REQPENDING:          {ASM has not responded yet}
         begin
          PassReqInfoToASM( RLREQREJECT );
          eventSelectEnable := TRUE;
         end;
      REQCLEARED:          {ASM changed its mind}
         begin
          PassReqInfoToASM( RLREQCOMPLETE );
          eventSelectEnable := TRUE;
         end;
      ALERTREQ:
         begin
          SetupForRequest;
```

```
      if activeCallState <> INCALLPROCSTATE then
        begin
         PassReqInfoToASM( RLREQREJECT );
         eventSelectEnable := TRUE;
        end
      else
        begin
         StopTimer;
          SendMsg( ALERT, EPMMoreElFromASM );
          if EPMMoreElFromASM then
            requestState := HANDOFFTOASM
          else
            begin
             PassReqInfoToASM( RLREQCOMPLETE );
             eventSelectEnable := TRUE;
            end;
        end;
    end;
  CLEARREQ:
    begin
     SetupForRequest;
     case activeCallState of
        NULLSTATE:
          begin
           PassReqInfoToASM( RLREQREJECT );
           eventSelectEnable := TRUE;
          end;
        OUTCALLPROCSTATE:
          begin
           SendMsg( DISC, TRUE );
           requestState := HANDOFFTOASM;
           ResetTimer;
           NewCallState( DISCREQSTATE );
          end;
        INCALLPROCSTATE:
          begin
           SendMsg( RELCOMP, TRUE );
           requestState := HANDOFFTOASM;
           StopTimer;
           NewCallState( NULLSTATE );
          end;
        CONNREQSTATE:
          begin
           SendMsg( DISC, TRUE );
           requestState := HANDOFFTOASM;
           ResetTimer;
           NewCallState( DISCREQSTATE );
          end;
        ACTIVESTATE:
          begin
           SendMsg( DISC, TRUE );
           requestState := HANDOFFTOASM;
           ResetTimer;
           NewCallState( DISCREQSTATE );
          end;
        DISCREQSTATE:
          begin
           SendMsg( RELEASE, TRUE );
           requestState := HANDOFFTOASM;
           ResetTimer;
           NewCallState( RELREQSTATE );
          end;
        DISCINDICATIONSTATE:
          begin
```

```
            SendMsg( RELEASE, TRUE );
            requestState := HANDOFFTOASM;
            ResetTimer;
            NewCallState( RELREQSTATE );
           end;
        RELREQSTATE:
           begin
           StopTimer;
            NewCallState( NULLSTATE );
            PassReqInfoToASM( RLREQCOMPLETE );
            eventSelectEnable := TRUE;
           end;
        end; (case)
    end;
CONNREQ:
    begin
     if EPMCallRefIn = 0 then  (new call request)
       begin
        activeCallRef       := GetFreeCallRef;
        activeCallState     := NULLSTATE;
        if activeCallRef = 0 then  (no free callRefs)
           begin
            PassReqInfoToASM( RLREQREJECT );
            eventSelectEnable := TRUE;
           end
        else
           begin
            SendMsg( SETUP, TRUE );
            requestState := HANDOFFTOASM;
            ResetTimer;
            NewCallState( OUTCALLPROCSTATE );
           end;
       end
     else
       begin
        SetupForRequest;
         if activeCallState <> INCALLPROCSTATE then
           begin
            PassReqInfoToASM( RLREQREJECT );
            eventSelectEnable := TRUE;
           end
        else
           begin
            SendMsg( CONNECT, TRUE );
            requestState := HANDOFFTOASM;
            ResetTimer;
            NewCallState( CONNREQSTATE );
           end;
       end;
    end;
DELAYREQ:
    begin
     SetupForRequest;
      if activeCallState <> INCALLPROCSTATE then
       begin
        PassReqInfoToASM( RLREQREJECT );
        eventSelectEnable := TRUE;
       end
      else
       begin
        ResetTimer;
         SendMsg( CALLPROC, EPMMoreElFromASM );
         if EPMMoreElFromASM then
            requestState := HANDOFFTOASM
```

```
            else
               begin
                PassReqInfoToASM( RLREQCOMPLETE );
                eventSelectEnable := TRUE;
               end;
            end;
         end;
      USERINFOREQ:
         begin
          SetupForRequest;
          if activeCallState <> ACTIVESTATE then
            begin
             PassReqInfoToASM( RLREQREJECT );
             eventSelectEnable := TRUE;
            end
          else
            begin
             SendMsg( USERINFO, TRUE );
             requestState := HANDOFFTOASM;
            end;
         end;
      end; {case}
   end
else if txFifoEmpty then  {the requestState is HANDOFFTOASM)
   begin
    txFifoOutputEnableBuf := FALSE;  {turn off the txFifo's output}
    PassReqInfoToASM( RLCLEARTOSEND );{tell ASM to output rest of msg}
    eventSelectEnable := TRUE;
   end;
end;

{ ******************************************************* }
{ ******************************************************* }

procedure TimeoutProcessing;

{ ********* nested routines ********** }

procedure NotifyASMOfTimeout;
begin
EPMTOutLineBuf    := TLTOUT;
EPMCallRefOutBuf  := activeCallRef;
EPMStateBuf       := activeCallState;
end;

procedure SendToutMsg( mt : VALIDMSGTYPE );
begin
txFifoPushBuf          := TRUE;
txFifoMsgBuf           := mt;
txFifoCauseBuf         := 111;
txFifoCallRefBuf       := activeCallRef;
txFifoElToFollowBuf    := FALSE;

{The following are dummy assignments}
txFifoDiagBuf          := 0;
txFifoCallStateBuf     := 0;
end;

{ *********  TimeoutProcessing ********** }

begin
activeCallRef := timeoutCallRef;
activeCallState := callRefTable[activeCallRef].callState;
case activeCallState of
```

```
      NULLSTATE:
         begin
          (this should never occur)
          StopTimer;
         end;
      OUTCALLPROCSTATE:
         begin
          SendToutMsg(DISC);
          ResetTimer;
          NewCallState( DISCREQSTATE );
         end;
      INCALLPROCSTATE:
         begin
          SendToutMsg(RELCOMP);
          ResetTimer;
          NewCallState( NULLSTATE );
         end;
      CONNREQSTATE:
         begin
          SendToutMsg(DISC);
          ResetTimer;
          NewCallState( DISCREQSTATE );
         end;
      ACTIVESTATE:
         begin
          (this should never occur)
          StopTimer;
         end;
      DISCREQSTATE:
         begin
          SendToutMsg(RELEASE);
          ResetTimer;
          NewCallState( RELREQSTATE );
         end;
      DISCINDICATIONSTATE:
         begin
          SendToutMsg(RELEASE);
          ResetTimer;
          NewCallState( RELREQSTATE );
         end;
      RELREQSTATE:
         begin
          StopTimer;
          NewCallState( NULLSTATE ); (Null implies released CallRef)
         end;
    end; (case)
  NotifyASMOfTimeout;
  eventSelectEnable := TRUE;
  end;

  { *************** Procedures exported by the interface ************ }

  procedure ProcessEPM;

  begin
  rxFifoPopDataBuf        := FALSE;
  txFifoPushBuf           := FALSE;
  EPMReqCtrlLineBuf       := RLNOREQ;
  EPMTOutLineBuf          := TLNOTOUT;
  EPMMsgLineBuf           := MLNOMSG;
  EPMElmLineBuf           := ELNOELM;

  (Check to see if ASM is finished sending. If so, then re-enable tx fifo)
  if NOT txFifoOutputEnable then
```

```
    txFifoOutputEnableBuf := EPMEventRequest = REQCLEARED;

  if eventSelectEnable then
     begin
      SelectActiveEvent;
      eventSelectEnable := activeEvent = NOEVENT;
     end;
  case activeEvent of
     NOEVENT:
        begin
        end;
     INCOMINGMSG:
        begin
         if rxFifoDataReady then
            MsgProcessing;
        end;
     INTERNALREQUEST:
        begin
         RequestProcessing;
        end;
     TIMEOUT:
        begin
         TimeoutProcessing;
        end;
    end; (case)
  end;

  { ********************************************************* }

  procedure OutputEPM;

  begin
  UpdateTimers;

  rxFifoPopData        := rxFifoPopDataBuf;
  rxFifoReset          := rxFifoResetBuf;

  txFifoOutputEnable  := txFifoOutputEnableBuf;
  txFifoReset          := txFifoResetBuf;
  txFifoPush           := txFifoPushBuf;
  if txFifoPushBuf then
     begin
     txFifoMsg          := txFifoMsgBuf;
     txFifoCause        := txFifoCauseBuf;
     txFifoDiag         := txFifoDiagBuf;
     txFifoCallRef      := txFifoCallRefBuf;
     txFifoElToFollow   := txFifoElToFollowBuf;
     txFifoCallState    := txFifoCallStateBuf;
     end;

  EPMReqCtrlLine  := EPMReqCtrlLineBuf;
  EPMTOutLine     := EPMTOutLineBuf;
  EPMMsgLine      := EPMMsgLineBuf;
  EPMElmLine      := EPMElmLineBuf;
  if (EPMReqCtrlLine <> RLNOREQ) or (EPMTOutLine <> TLNOTOUT) then
     begin
      EPMCallRefOut  := EPMCallRefOutBuf;
     EPMState        := EPMStateBuf;
     end;
  if EPMMsgLine <> MLNOMSG then
     begin
     EPMMsgType      := EPMMsgTypeBuf;
      EPMCallRefOut  := EPMCallRefOutBuf;
     EPMState        := EPMStateBuf;
```

```
    end;
  if EPMElmLine <> ELNOELM then
    begin
     EPMMsgOctet     := EPMMsgOctetBuf;
     EPMElementType  := EPMElementTypeBuf;
    end;
end;

{ ********************** Module initialization code ************* }

begin
InitEPM;
end.
```

Appendix E

Glossary

CCITT Defined Terms:

The following terms are defined in this section:

12) Access protocol
6) Bearer service
4) Circuit
7) Connection
11) Interface
5) Link
9) Network
11) Network-user protocol
13) SDL
1) Signal
8) Signaling
2) Transmission
3) Transmission channel
5) Transmission link
10) User

The following definitions are either exactly as defined by the CCITT Red Book or expanded slightly for clarity:

1) **signal**
   A physical phenomenon one or more of whose characteristics may vary to represent information.

2) **transmission**
   The action of conveying signals from one point to one or more other points.

3) **channel, transmission channel**
   A means of unidirectional transmission of signals between two points. Often channel is used to specifically refer to a particular time slot or bandwidth capability of a time division multiplex of several channels that comprise a transmission link.

4) **circuit**

A combination of two transmission channels permitting bi-directional transmission of signals between two points to support a single communication. If the communication is by nature unidirectional the term circuit is sometimes used to designate the single channel providing the facility. When the term channel is interpreted in its more common form as a particular time slot on a link, then a circuit can actually be comprised of several channels in each direction.

5) **link, transmission link**

A means of transmission with specific characteristics between two points.

6) **bearer service**

A type of telecommunication service that provides the capability for the transmission of signals between link end points or more generally between users of the network.

7) **connection**

A concatenation of transmission channels or circuits, switching and other functions set up to provide a bearer service.

8) **signalling**

The exchange of information concerned with the establishment and control of connections.

9) **network**

A set of nodes and links that provides connection between two or more defined points to facilitate telecommunication between them.

10) **user**

A person or machine delegated by a customer to use the services and/or facilities of a telecommunication network.

11) **interface**

The common boundary between two equipments.

12) **access protocol, network-user protocol**

A defined set of procedures that is adopted at an interface to enable the user to employ the services and/or facilities of the network.

13) **SDL**

Functional Specification and Description Language. A CCITT standard for protocols.

Terms used in the thesis:

active call

A call is considered to be active when the user channels are available to support the requested communication service. In other words, the call is active once its service has been established.

Application Specific Module (ASM)

Depending on the context, the ASM is either a logical module of the protocol processor model or the implementation of that module. The ASM handles the application specific functions of the network layer protocol.

application

As used in the thesis, the term is synonymous with "ISDN communication equipment". It should not be confused with the application layer of the OSI model.

basic rate interface

Also called 2B+D. It is the ISDN access protocol specified for the standard 2-wire twisted pair local loop.

Bearer Capability Information element

This information element is used to specify the end-to-end characteristics of the service that the user wishes to establish.

bearer channel(s)

The channel(s) or time slot(s) that have been allocated to provide the bearer service of a call.

broadband ISDN (B-ISDN)

The ISDN access protocol for high bandwidth mediums such as fiber optics.

call

The service of providing a (usually circuit switched) connection between users.

Call State Information element

This information element is used by the Status message to convey the message originator's call state. The call to which the call state applies corresponds to the message's call reference.

call control procedures

The steps that the signalling system must take to establish, manage, and disestablish calls. Call control procedures include, but are not limited to, the exchange of signalling messages with the network layer at the other end of the transmission link.

call reference

A unique one or two byte numeric "tag" that is assigned to every call by the calling party. The call reference is included in every signalling message to

associate the messages with the call to which it applies. Call Reference assignments have only local (i.e. link) significance; they do not apply to the call on an end-to-end basis. The high order bit of the call reference is set to 0 by the calling side and 1 by the called side. In this way call reference assignment collision is avoided.

call suspension

A feature by which the user is permitted to temporarily disconnect communication equipment for the purpose of moving it (in a multi-drop configuration) or changing it. The network keeps the call "on hold" until the user re-connects using a Resume signalling message.

Cause information element

This information element is mainly used in two situations: 1) in Disconnect, Release and Release Complete messages to convey the reason for disestablishment, and 2) in Status messages to convey the error condition which caused the Status message to be generated.

Channel identification information element

This information element is used to convey the channel number of the bearer channel(s) to be used to support the call. The channel numbers have only local (i.e. link) significance.

channel identification, channel number

The physical time slot number (in relation to some standard framing on the link) of the channel.

common channel signalling (CCS)

A technique of call control signalling (usually transmitted over a separate channel) in which multiple user channels are controlled over a single signalling channel. The distinguishing feature of CCS is that the call control information must be multiplexed (usually statistically) onto the single channel.

diagnostic

A field within the Cause information element used to convey additional information in the Status message. Normally the diagnostic is the call reference, message type or element type that has been found to be in error.

element type

An eight bit tag assigned every information element type which uniquely identifies the type. The first byte of every information element contains this tag.

end of a link

The equipment terminating one end of a transmission link. Also used to refer to the signalling system's protocol processor within the equipment.

Event Processing Module (EPM)

A sub-module of the PSM. The term normally refers to the implementation of the module and/or the functions it performs.

event

Also called a signalling event, it is the occurrence of a stimulas to the signalling system, where stimulas implies the required response may modify the overall status of a call. In general a signalling event can be the arrival of an incoming signalling message on the D channel, a timeout occurring, or an internal request. Events are normally associated with a single call reference.

FIFO

A First In-First Out buffer or queue.

forward error correction (FEC)

The technique of adding additional information to a transmitted signal in a such a fashion as to permit the receiver to decode the additional information and correct certain types of errors that may have occurred during the process of transmission.

handshaking

Information exchange based on previously agreed upon procedures or conventions. The term is somewhat synonymous with "signalling".

head (of a FIFO)

The next element of the FIFO to be made available when the FIFO is read or "popped".

I.451

The CCITT recommendation for the ISDN signalling system's network layer. It is identical to the Q.931 recommendation.

Incremental development

The technique of software development by which the system is initially built to "run" using functionally incomplete sub-modules which are then fleshed out using incremental advances.

Information element

A grouping of fields of signalling information into a single entity. It is composed of an information element type and a length field followed by the other fields.

internal request

A signalling event that is initiated locally, where locally means by the communication equipment itself.

LAPB

The data link layer protocol for X.25 type networks.

LAPD

The ISDN data link layer protocol for basic and primary rate interfaces.

local loop

The 2-wire twisted pair cable plant that connects the subscriber (user) to the local switching office of the telecommunication network.

mandatory information element

An information element that must be present in a signalling message for that message to be considered valid.

message type

A single byte field that uniquely identifies the signalling message's type. It is a mandatory part of every message which always follows the call reference.

network layer (protocol) processor

The module of the signalling system's protocol processor that is responsible for processing the network layer signalling protocol.

network side

The switch side of transmission link. Is usually only applicable to star topologies.

octet

The CCITT term for a byte: 8 bits.

operating company

As used in the thesis, the private or public organization or entity responsible for (at least) the access portion of the telecommunications network. The meaning of the term varies depending on the overall structure of the country's telecommunication network.

optional information element

An information element that may be present in a signalling message, but can also be absent without causing the message to be classified as errored. Some CCITT optional elements may actually be considered mandatory by local implementations.

pipelined processing

The technique of information or signal processing by which the entity to be processed passes through successive stages, and where at any one time each stage may be processing a different entity.

primary rate interface

The ISDN access protocol based on a physical layer compatible with the first level of the digital transmission hierarchy. Also called 23B+D for the North American hierarchy and 30B+D for the CEPT hierarchy.

Protocol Specific Module (PSM)

Depending on the context, the PSM is either a logical module of the protocol processor model or the implementation of that module. The PSM handles the protocol specific functions of the network layer protocol.

protocol discriminator

A first byte of every I.451 signalling message, it identifies the protocol used to structure the message. I.451 based messages use 00001000.

protocol processor

Also signalling system's protocol processor. The processing entity responsible for implementing the signalling system for a the communication equipment at the end of a transmission link.

rapid prototyping

The technique of software development by which the system requirements evolve with the system development. This is done by providing the user with successively improved prototypes which he uses to evaluate the system and his own requirements, and then recommends changes.

Red Book

The 1985 publication of CCITT recommendations. The next publication - the Blue Book - is due in 1988.

RS-232

Also RS-232-C. An Electronics Industries Association (EIA) physical layer standard that supports asynchronous, serial, full duplex communications of up to 19.2 Kb/s. The standard defines two types of equipment, Data Terminal Equipment (DTE) and Data Communications Equipment (DCE). The standard assumes DTE-DCE connections will be made, although DCE-DCE and DTE-DTE connections can be creating using a "null modem" between them. By the way, RS denotes "recommended standard".

signalling message

A network layer generated collection of signalling information that is passed to the data link layer and sent as a packet.

stimulus terminal

A piece of communication equipment (normally on the customer premise and classified as user side equipment) that does not remember its own call state. It is a slave to the network side.

stream processing

The technique of message processing where the message is processed by the various layers of the protocol processor in a pipelined fashion, normally on a byte-by-byte basis. This is contrasted by the common technique of passing whole messages from one layer to another.

symmetric protocol

An interface protocol designed such that the communication equipment can successfully establish service connections without prior assignment or negotiation of "roles" such as DTE-DCE, or user side - network side.

tail (of a FIFO)

The location where the next element will be written or pushed into the FIFO.

terminal, terminal equipment

Equipment that provides the user with access to the services and/or facilities provided by the network, and that provides the functions necessary for the operation of the access protocol by the user. When discussing symmetric protocols, terminal is used to mean the equipment that interfaces the user to the transmission link, regardless of whether or not the other end of the link is a telecommunications network, a LAN, a computer, or another terminal.

Time-Slot Interchange (TSI) switch

As used in the thesis, a TSI is a type of memory element that stands between two continuous, unidirectional time division multiplexed signals. A common implementation has the incoming signal cyclically writing its channels into memory locations where the write address equals the channel number, while the outgoing signal reads its channels from memory locations with the read address is controlled by a connection memory, which is simply a dynamic mapping between memory locations and the outgoing signal's TDM channels. In this way, and with only a single frame delay, any incoming channel can be written into any outgoing channel under the control of the connection memory.

timeout

An internal signalling event associated with a single call reference denoting that the timer for that call has expired.

two-way independent channel selection

A technique of selecting bearer channels in a symmetric protocol whereby each end of the link is master in the transmit direction and slave in the receive direction. In this way bearer channel selection collisions are avoided.

user channel

The bearer channel(s) available to the user for the service requested. User channels are often called B channels, although this is misleading since B channels are strictly 64 Kb/s channels. There are user channels (H0, H1, etc.) that are not B channels.

user information

Signalling information that only has relevance to the users on an end-to-end basis. The network's signalling system acts solely as a carriage for these signals, it does not attempt to interpret them.

user side

The user's end of a standard (non-symmetric) ISDN link.